Practically Efficient Secure Small Party Computation over the Internet

A THESIS

SUBMITTED FOR THE DEGREE OF

Master of Technology (Research)

IN THE

Faculty of Engineering

BY

Megha Byali

Computer Science and Automation

Indian Institute of Science

Bangalore – 560 012 (INDIA)

November, 2019

Declaration of Originality

I, **Megha Byali**, with SR No. **04-04-00-10-22-16-1-13901** hereby declare that the material presented in the thesis titled

Practically Efficient Secure Small Party Computation over the Internet

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2016-2019**.
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: 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: Advisor Signature

DEDICATED TO

My beloved Parents and loving Brother

who stood by me in every phase of my life

Thank you for being there

Acknowledgements

Excellence is about desire: "I'll not let a single ball go past me" ; "Hit one more to me".

-Harsha Bhogle

First and foremost, I would like to extend immense gratitude to my research advisor Dr. Arpita Patra, for accepting me as part of her lab and introducing me to the world of "Secure Computation". I still remember the time when she took a leap of faith and accepted me as her research student despite my lack of understanding of cryptography to begin with. Ever since, she has played an enormous role in identifying and honing my skills. I'm forever indebted to her for bestowing all the opportunities that have come my way. I've grown as a researcher and moreover as a person under her guidance and support. Despite having personal commitments, the dedication that she has offered while working with me in all the times we've submitted papers has left me in awe. She has stayed up late with me during submissions and has meticulously helped me in all our research discussions and writing. We have had meaningful conversations about various topics on several occasions and it has led me to appreciate the enthusiasm she brings on the table. Her perseverance and inquisitiveness are the virtues I wish to inculcate within me. In summary, it has been a wonderful experience to work with her.

Besides my advisor, I would also like to express my sincere gratitude to Dr. Carmit Hazay for inviting me to *Bar-Ilan University, Israel* and hosting me as a research collaborator. She gave me the opportunity to meet with some of the pioneer researchers at *Bar-Ilan University.* It was during the time with her that I started working on my thesis and she's been a great mentor. She made my stay in Israel a very comfortable one and was helpful in the numerous discussions that we've had over time. She was pivotal in giving me a different perspective about research and it has been a pleasure working with her. The time I spent in Israel has been one of the most memorable times in recent years.

I would also like to thank my collaborators who were also my lab mates: Ajith Suresh, Divya Ravi and Pratik Sarkar for being great researchers to work with. Our unending discussions, be it research or gossip have been pleasurable. I would rather consider them more as friends than lab mates. I've had memorable times with them specially at Denmark. My special regards to Divya for being a great mentor and helping me in every step of the way during my research. It has been a pleasant experience to have her as a co-author for my first conference paper.

My co-author for the papers based on this thesis, Swati Singla has been amazing to work with. Our chemistry as co-authors has been a remarkable one. The resemblance that we've shared in our career choices is uncanny. She has also been one of my great friends at IISc, somebody I could always rely on. We've shared some great moments, be it in IISc, Israel or Denmark. I appreciate her for being there in my difficult situations, for being my personal photographer and bearing my constant cribbing! which I'm sure she has loved.

My warm regards specially to Ajith Suresh for helping me out with all the coding conundrums!. My lab-mates Arun Joseph, Nishat Koti and Harsh Choudhary deserve special acknowledgements. They have made learning, a fun experience. I would also like to thank my friend, Shivika Narang for being a constant support. I wish to extend my love to all friends at IISc for making my stay a memorable one.

My hearty gratitude goes out my friend Deepti Upadhya with whom I've shared countless laughter and moments worth living. Her sarcasm, advice and positiveness has made me a better person. We've been friends for as long as I can remember and everything is complete when she is around. My warm regards to my friend Anant Nayak for being my emotional support and staying by me. My love goes out to my wonderful friends Supriya Doddagoudar and Trishla Kalal for being with me through thick and thin. I've cherished all the fun, sarcasm and philosophical conversations that we've shared overtime.

This acknowledgement is incomplete without the mention of my parents Ashokraj and Kalpana, to whom my truest gratitude goes out. Their unending love and support in every step of my life has made me who I'm today and I will forever be grateful to them for raising me as an independent person. They've bestowed upon me with everything I could wish for and have been great teachers of my life. My mother has been the greatest friend and teacher I've ever known. She's always taught me to dream big and I would consider myself blessed if I could be at least half the woman she is. My father has taught me to be ambitious and self reliant. My unending love goes out to my brother Sagar, who has had immense influence over me. He has been my go to person throughout life. His philosophical thoughts, sarcasm and advice are worth living for. I always cherish the fun and fights we've had over time. The amount of learning I've had being with him is enormous.

Lastly, I would like to thank IISc, IACR, ACM CCS'18 and TPMPC'18 for granting me funds to attend conferences.

Abstract

Secure Multi-party Computation (MPC) with small population has drawn focus specifically due to customization in techniques and resulting efficiency that the constructions can offer. Practically efficient constructions have been witnessed in the setting of both honest majority and dishonest majority. In this work, we investigate the efficiency of a wide range of security notions in the small party domain with 5 parties and 4 parties. Being constant-round, our protocols are best suited for real-time, high latency networks such as the Internet. All our constructions are backed with experimental results.

In the setting of five parties with honest majority, we present efficient constructions with unanimous abort (where either all honest parties obtain the output or none of them do) and fairness (where the adversary obtains its output only if all honest parties also receive it) in a minimal setting of pairwise-private channels. With the presence of an additional broadcast channel (known to be necessary), we present a construction with the strongest security of guaranteed output delivery (where any adversarial behaviour cannot prevent the honest parties from receiving the output). The broadcast communication is minimal and independent of circuit size. In terms of performance (communication and run time), our protocols incur minimal overhead over the best known selective abort protocol of Chandran et al. (ACM CCS 2016) while retaining their round complexity. Further, our protocols for fairness and unanimous abort can be extended to n-parties with at most $\sqrt{n}$ corruptions, similar to Chandran et al.

In the setting of four parties, surpassing the traditional honest majority model, we achieve stronger security goals in a mixed model where minority of the parties are actively corrupt and additionally some parties are passively corrupt, thus giving an overall dishonest majority. We present the first efficient constructions that tolerate a mixed adversary corrupting 1 party actively and 1 party passively and achieve the security goals of guaranteed output delivery and fairness. Our constructions adhere to the feasibility result of Hirt et al. (CRYPTO'13).

Going beyond the most popular honest-majority setting of three parties with one corruption, our results demonstrate feasibility of attaining stronger security notions at an expense not too far from the least desired security of selective abort.

Publications based on this Thesis

- **Megha Byali**, Carmit Hazay, Arpita Patra and Swati Singla. *Fast Actively-secure Five Party Computation with Security Beyond Abort.* **ACM CCS 2019**.
- **Megha Byali**, Arpita Patra, Divya Ravi and Swati Singla. *Beyond Honest Majority: On the Efficiency of 4-Party Computation in High-latency Networks.* Under Submission.

Other Publications

- **Megha Byali**, Arun Joseph, Arpita Patra and Divya Ravi. *Fast Secure Computation for Small Population over the Internet.* **ACM CCS 2018**.
- **Megha Byali**, Pankaj Dayama, Shivika Narang, Yadatti Narahari and Vinayaka Pandit. *Trusted B2B Market Platforms using Permissioned Blockchains and Game Theory.* **IEEE Conference on Blockchain and Cryptocurrency**.
- **Megha Byali**, Nishat Koti, Arpita Patra, Divya Ravi and Swati Singla. *Speedo4: High-Speed Secure 4-Party Computation over the Internet.* Under Submission.
- **Megha Byali**, Harsh Chaudhari, Arpita Patra and Ajith Suresh. *FLASH: Fast Maliciously Secure 4PC Framework for Machine Learning.* Under Submission.
- **Megha Byali**, Arpita Patra, Divya Ravi and Pratik Sarkar. *Efficient, Round-optimal, Composable Oblivious Transfer and Commitment Scheme with Adaptive Security.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Secure Multiparty Computation (MPC) [Yao82, GMW87, CDG87] is an area of cryptography that has evolved breathtakingly over the years in its attempt to secure data while computing on it. MPC focuses on the problem of enabling a set of n mutually distrusting parties to perform joint computation on their private inputs in a way that no coalition of t parties can affect the output of computation or learn any additional information beyond what is revealed by the output. In other words, MPC guarantees correctness of computation and privacy of inputs. The literature of MPC has witnessed plethora of works from a theoretical standpoint, however, the focus on building practice-oriented MPC [DPSZ12a, WRK17, BHKL18] constructs has gained momentum only in the recent years owing to the rising demand for efficiency in real-time networks such as the Internet. Based on the corruption threshold, the vast literature of MPC is traditionally categorized into dishonest majority [GMW87, DO10, BDOZ11, DPSZ12b, AJL$^+$12, NNOB12, LPSY15, WRK17] and honest majority [BGW88, RB89, BMR90, DN07, BH07, BH08, BFO12, MRZ15]. While both have received attention in the efficiency studies, designing practical MPC with honest majority is a captivating area of research [MRZ15, AFL$^+$16, FLNW17, CGMV17, PR18, BJPR18] for the various reasons illustrated below.

The paramount benefit of having honest majority enables the computation to achieve stronger security goals such as *fairness* (adversary obtains output if and only if all honest parties do) and *guaranteed output delivery* (GOD) (any adversarial behaviour cannot prevent the honest parties from receiving the output) [Cle86]. These properties are desirable in real-life owing to limited time and resource availability, as they bind the parties to participate in the computation and thus keep the adversarial behaviour in check. Furthermore, lack of such strong guarantees can be detrimental in practice. For instance, in real-time applications such as e-commerce and e-auction, an adversary can always cause an abort if the outcome is not in its favour unless a stronger security notion is ensured. In e-voting, the adversary can abort the computation repeatedly, yet learn the outputs each time and use them to rig the election. Apart from enabling stronger security goals, honest-majority allows design of efficient protocols solely using symmetric-key functions. For instance, the necessity of a public-key primitive for realizing oblivious transfer can be replaced with symmetric-key primitives, as exhibited by our protocols and [CGMV17]. Further, this setting enables design of information-theoretic protocols [BGW88, RB89, BFO12, IKKP15], besides the computational ones. Thus, these strong notions have driven a lot of research. To elaborate, [DI05, DI06] show constant-round protocols with GOD. The round-optimality of these notions have been studied in [GIKR02, GLS15, PR18] and 3 rounds is proven to be necessary. Lately, round-optimal MPC protocols with GOD appeared in [GLS15, ACGJ18, BJMS18] relying on either Common Reference String (CRS) or public-key operations, in [ACGJ19, ABT19] under super-honest-majority $t < n/4$ and in [PR18] for the special case of 3-party solely from symmetric-key primitives. The work of [DOS18] shows how to compile honest majority MPC protocol for arithmetic circuits with abort (and several other constraints) into a protocol with fairness while preserving its efficiency. Interestingly, while [Cle86] rules out fairness in dishonest majority, [BK14, ADMM14, CGJ+17, PST17] demonstrate its feasibility relying on non-standard techniques such public bulletin boards, secure processors or penalties (via Bitcoin).

Another widely acceptable demarcation of the protocols apart from the traditional honest majority and dishonest majority is in terms of the power of adversary; which can be *active* (parties deviate arbitrarily from the protocol) or *passive* (the protocol steps are correctly followed but the parties can gossip to glean additional information). The work of [Cha89, DDWY93, FHM98, HMZ08] overcomes this strict partition and considers the notion of *mixed* adversary who can selectively corrupt some parties to be active and some additional parties to be passive. Such protocols are more suitable for practical scenarios where the adversary may have wider range of corruption options, and is not necessarily restricted to purely active or passive. This model is particularly preferable for critical systems of financial data analysis [BTW12], secure auctions [DGK09], federated learning and prediction [MR18], voting [KMO01, NBK15] and secure aggregation [BIK+17] where input privacy is of paramount importance and yet, a robust computation (to the extent theoretically feasible) is desirable. In this direction, we present the first efficient constructions in the four-party (4PC) setting, against a *mixed* adversary corrupting one party actively and one party passively.

Since inception, the primary focus of MPC has been on generic constructions with n parties. Yet, the regime of practical MPC has seen major breakthroughs in the small-party domain: 3-5. Real-time applications such as Danish Sugar-Beet Auction [BCD+09], statistical and financial data analysis [BTW12], email filtering [LADM14], distributed credential encryption [MRZ15], Kerberos [AFL+16], privacy-preserving machine learning [MRSV17], efficient MPC-frameworks such as VIFF [Gei07], Sharemind [BLW08] and ABY-Arithmetic Boolean Yao [MR18] are crafted for 3 parties with one corruption. The setting of 4, 5 parties with minority corruption has been explored in [CGMV17, IKKP15, BJPR18]. The most popular setting of 3/4 parties with 1 active corruption brings to the table some eloquent custom-made tools such as the use of Yao's garbled circuits [Yao82] to achieve malicious security [MRZ15, PR18, BJPR18], spending just 2-3 elements per party in arithmetic circuits [ABF+17] and sure-election of one honest party as a trusted party in case the adversary strikes [BJPR18, PR18]. These techniques rely on the adversary not having an accomplice to cause damage. However, the moment adversary has a collaborator (2 corruptions), these custom-made tools fall apart, thus elevating the challenge of achieving desired security with real-time efficiency. In this thesis we consider,

(i) *Honest Majority model*– Efficient MPC for 5 parties (5PC) with 2 corruptions and treat it with securities of unanimous abort (where either all honest parties obtain the output or none of them do), fairness and GOD, at an expense that is not too far from the result of [CGMV17] achieving least desired security of selective abort (the adversary on receiving the output can arbitrarily choose which of the honest parties get the output).

(ii) *Mixed Adversary model*– Efficient MPC for 4 parties (4PC), first of their kind, with simultaneous 1 active and 1 passive corruptions that promise fairness and GOD. Note that, the work of [Cle86] shows that the security notions of fairness / GOD can be achieved under at most an active *minority*. However, in this adversarial model, we aim to provide strong security while going beyond strict honest majority and considering an additional (purely) passive party (apart from active minority) shown to be feasible in [HLM13]. Specifically, we consider only one passive party as opposed to 2 in 4PC, owing to the feasibility threshold of [HLM13] which introduces a *dynamic trade-off* between active and passive corruptions. In particular, [HLM13] shows that the stronger goals of fairness and GOD are attainable when $2t_a + t_p < n$ where t_a denotes active corruptions and t_p denotes the (purely) passive corruptions. This directly rules out the possibility of fair protocols with 1 active and 2 additional passive corruptions in the 4-party setting; implying our setting of one active and one passive corruption is optimal for fair / GOD protocols.

1.1 Literature

The notable works on MPC for small parties come in two flavours– low-latency and high-throughput protocols. Relying on garbled circuits, the former offers constant-round protocols that serve better in high-latency networks such as the Internet. The latter, built on secret sharing tools, aim for low communication (bandwidth), but at the cost of rounds proportional to the depth of the circuit representing the desired function. These primarily cater to low-latency networks. We focus on the former category in our work. As efficiency studies considering mixed adversary is limited and no relevant literature exists for small party domain to the best of our knowledge, we mainly focus on MPC with small population considering the traditional honest and dishonest majority below.

The work most relevant to ours (in both 5PC and 4PC) is [CGMV17] that proposes a 5PC protocol achieving the weak notion of selective abort against two malicious corruptions. Their customization for 5PC resulted in an efficient protocol for actively-secure distributed garbling of 4 parties, relying solely on the passively-secure scheme of [BLO16], saving 60% communication than [BLO16] with four corruptions. In the 3-party (3PC), 4-party (4PC) domain, [MRZ15, IKKP15] gave a 3PC with selective abort. [IKKP15] also gave a 2-round 4PC with GOD. Recently, [BJPR18] improved the state-of-the-art with efficient 3PC and 4PC achieving fairness and GOD with minimal overhead over [MRZ15]. In the dishonest-majority setting, the protocol of [CKMZ14] studies 3PC with two active corruptions. Orthogonally, recent works [AFL+16, ABF+17, FLNW17, CCPS19, EOP+19] in the high-throughput setting with non-constant rounds, show abort security in 3PC with one corruption. The works of [CGH+18, NV18, DOS18, CCPS19] additionally include constructs attaining fairness. The recent work of [GRW18] explores the 4-party setting with one malicious corruption and considers the stronger security notions of fairness and GOD.

1.2 Our Contribution

In the regime of low-latency protocols which is of interest to us, the widely known works [MRZ15, IKKP15, CGMV17], despite being in honest majority, trade efficiency for security and settle for weaker guarantees such as selective abort. With $3, 4$ parties, [IKKP15, PR18, BJPR18] demonstrate that fairness, GOD are feasible goals and present protocols with minimal overhead over those achieving weaker notions. Our work is yet another attempt in this direction, focused on the 4-party and 5-party setting. Being efficient and constant-round, our protocols are best suited for high latency networks such as the Internet. Designed in the Boolean world, our protocols are built on the semi-honest variant of the distributed garbling scheme of [WRK17] while leveraging the techniques of seed distribution and Attested Oblivious Transfer of [CGMV17]. The semi-honest variant of the distributed garbling scheme of [WRK17] is superior to the state-of-the-art semi-honest distributed garbling scheme of [BLO16]. The generality of our protocols is such that they can accommodate any passively secure distributed garbling scheme as a building block. Our theoretical findings are backed with implementation results with the choice of benchmark circuits AES-128 and SHA-256. Below we summarize our contributions.

In the traditional honest majority model, we present efficient, constant-round 5PC protocols tolerating two malicious corruptions that achieve security notions ranging from unanimous abort to GOD, solely relying on symmetric-key primitives.

5PC with Fairness and Unanimous Abort In a minimal network of pairwise-secure channels, we achieve fairness and unanimous abort in 5PC with performance almost on par with [CGMV17], all consuming 8 rounds. On a technical note, building on [CGMV17], we achieve fairness by ensuring a robust output computation phase even when the adversary chooses not to participate in the rest of the output computation on learning the output herself. This is realized using techniques which enforce that, in order to learn the output herself, the adversary must first aid at least one honest party compute the correct output. Further, we employ techniques to allow this honest party to release the output and convince about the correctness of the same to remaining honest parties. Our 5PC with unanimous abort is obtained by simplifying the fair construct such that the adversary can learn the output all by herself without any aid from honest parties, but if she helps at least one honest party get the output, then that honest party aids fellow honest parties to get the output (as in fair construct). Both our 5PC protocols with fairness and unanimous abort can be extended to n parties under the constraint of $t = \sqrt{n}$ corruptions which was established in [CGMV17].

5PC with GOD Our protocol uses point-to-point channels and a broadcast channel. The latter is inevitable as we use optimal threshold [CL14]. As broadcast is expensive in real-time, we limit broadcast communication to be minimal and primarily, independent of circuit, input and output size. Our implementation uses a software broadcast based on Dolev-Strong protocol [DS83]. On the technical side, our protocol relies on 2-robust techniques– 4-party 2-private replicated secret sharing (RSS) scheme for input distribution and seed-distribution of [CGMV17] to ensure each party's role is emulated by two other parties. These strategies ensure that each piece of intermediate data is with a 3-party committee and any wrong-doing by at most 2 parties will ensue conflict. When a conflict occurs, we determine a smaller instance of a 3PC with at most 1 corruption to compute the output robustly. Our technical innovations come from maintaining– (A) input privacy, while making two 3-party committees, one formed by RSS and one by seed-distribution, interact; (B) input consistency across the 3PC and outer 5PC. Due to the use of customized tools for small parties such as RSS, conflict identification and running a smaller 3PC instance, this protocol cannot be scaled to n-parties while retaining the goal of efficiency.

In the setting of *mixed model*, where the adversary can corrupt parties both actively and passively, we present two concrete 4PC constructions, against 1 active and 1 passive corruption ($t_a = t_p = 1$) achieving GOD and fairness.

4PC with GOD and Fairness Our protocols are highly efficient in nature due to the use of semi-honest primitives to begin with. The setting, though goes beyond the natural honest-majority, is able to leverage the techniques of passive distributed garbling, attested oblivious transfer and seed distribution (used in the face of two active corruptions among 5 parties in [CGMV17]), mainly due to the semi-honest nature of the second corrupt party.

On the technical side, for the 4PC GOD protocol, the prime innovations include– (1) Use of *two* 1-out-of-2 semi-honestly secure oblivious transfer (OT) [EGL85] to tackle a malicious corruption as opposed to *one* expensive maliciously secure OT for transfer of data and still preserve input privacy. (2) Identification and exclusion of two conflicted parties (one of which is guaranteed to be the actively corrupt) and leveraging a passive 2PC based on Yao's garbled circuit [Yao82] to complete the computation. (3) Measures to ensure input consistency and privacy throughout the computation. On the other hand, the 4PC fair protocol is a simplification of the 4PC GOD and we allow parties to abort before any party obtains the output since it is acceptable for the execution to abort in such case owing to the weaker security guarantee. The prime innovation involves ensuring the robust computation of output by honest parties once the corrupt evaluator has obtained the output. This is done by denying the evaluator of the output till the result of circuit evaluation is communicated by the evaluator. Moreover as in 4PC GOD protocol, semi-honestly secure OTs are used to improve efficiency.

Empirical Comparison. A consolidated view of our results is presented below outlining the security achieved, rounds used, use of broadcast (BC) and empirical values. The values indicate the overhead in maximum runtime latency in LAN, WAN and total communication (CC) over [CGMV17] that offers selective abort in 8 rounds. The range is composed over the choice of circuits: AES-128 and SHA-256 and the left value in the range corresponds to AES, while the right value indicates SHA. AES is a smaller circuit, with 33616 gates, compared to 236112 gates of SHA. (**(g)** for a value indicates gain over [CGMV17]. The worst case run of 5PC with GOD is calculated plugging in the state of the art robust 3PC [BJPR18] and the worst case 4PC GOD is calculated plugging in [Yao82] with the state of the art optimization of [ZRE15]).

Security	**BC**	**LAN** (ms)	**WAN** (s)	**CC** (MB)
unanimous abort	✗	0.65-2.87	0.2-0.01	0.16-0.09
5PC with fairness	✗	1.05-10.95	0.28-0.03	0.2-0.13
5PC with GOD (honest run)	✓ [CL14]	3.94-4.92	1.16-0.82	0.17-0.07
5PC with GOD (worst case)	✓ [CL14]	6.33-19.42	2.26-2.33	0.49-6.22
4PC with fairness	✗	2.93(**g**)-23.14(**g**)	0.37(**g**)-0.99(**g**)	12.83(**g**)-132.36(**g**)
4PC with GOD (honest run)	✗	2.54(**g**) -17.38(**g**)	0.01(**g**)-0.54(**g**)	12.77(**g**)-132.24(**g**)
4PC with GOD (worst case)	✗	1.14(**g**)-1.9(**g**)	0.23-0.29(**g**)	12.47(**g**)-129.24(**g**)

All protocols barring the ones with GOD maintain the same circuit-dependent communication as [CGMV17]. The GOD protocols cost two circuit-dependent communication, one in the outer protocol (5PC/4PC) and one in smaller instance (3PC/2PC). This is reflected in the cost of worst case run of our GOD protocols. For all other constructions in 5PC, the overhead comes from extra communication (commitments to be precise) that is dependent only on the input, output size. Since SHA is a bigger circuit, its absolute overheads for 5PC are more than AES in most cases but the percentage overheads are better for SHA than AES. The factor of additional communication overhead incurred by our 5PC protocols for SHA when compared to AES circuit is far less than the factor of increase in the total communication for SHA over AES in [CGMV17]. This indicates that the efficiency of our protocols improves for larger circuits. The saving for our 4PC protocols over [CGMV17] is due to the difference in the number of parties. Nevertheless, our 4PC protocols achieve stronger security of fairness and GOD while going beyond strict honest majority as opposed to the weakest security of selective abort achieved by [CGMV17] in honest majority.

1.3 Outline of this Thesis

We begin the thesis starts by introducing the basics of MPC and a high-level overview of the preliminaries most relevant to our work. This is followed by the protocols and their security proofs. We divide the thesis into two parts: first, 5PC with honest majority appearing in Chapters 4,5,6 and second, 4PC with mixed adversary appearing in Chapter 7,8. We now present the thesis outline.

- **Chapter 2:** This chapter begins with the discussion of the secuirty model, notations used and the formal functionalities of the security notions that are achieved in our work, followed by the quick overview of preliminary tools and primitives used throughout the thesis.

- **Chapter 3:** In this chapter, we describe in detail, the basic efficient building blocks, the distributed garbled circuit construction and evaluation technique for five party and four party protocols.

- **Chapter 4:** In this chapter, we begin with the honest majority model in 5PC and present our 5PC with fairness. We first give a technical overview of our 5PC with fairness protocol. We then move onto a formal description of our protocol, followed by a rigorous security proof.
- **Chapter 5:** In this chapter of 5PC with honest majority, we first give a technical overview of our 5PC with unanimous abort protocol. We then move onto a formal description of our protocol, followed by the security proof.
- **Chapter 6:** In this chapter of 5PC with honest majority, we first give a technical overview of our 5PC with guaranteed output delivery protocol. We then move onto a formal description of our protocol, followed by a rigorous security proof.
- **Chapter 7:** In this chapter, we begin with the mixed adversary in 4PC and present our 4PC with fairness. We first give a technical overview of our 4PC with fairness protocol. We then move onto a formal description of our protocol, followed by a rigorous security proof.
- **Chapter 8:** In this chapter of 4PC with mixed adversary, we first give a technical overview of our 4PC with guaranteed output delivery protocol. We then move onto a formal description of our protocol, followed by a rigorous security proof.
- **Chapter 9:** We discuss the efficiency of our 4PC and 5PC protocols compared to their respective state-of-the-art elaborately in this chapter.
- **Chapter 10:** We conclude with summary of the thesis and possible future directions to our work.

Chapter 2

Preliminaries

2.1 Security Model and Notations

We consider a set of 5 parties $\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5\}$, where each pair is connected by a pair-wise secure and authentic channel. The presence of a broadcast channel is assumed only for the 5PC GOD protocol where it is known to be necessary [CL14]. We model each party as a non-uniform probabilistic polynomial time (PPT) interactive Turing Machine. We consider a static security model with honest majority, where a PPT adversary $\mathcal{A}$ can corrupt at most 2 parties at the onset of protocol. Adversary $\mathcal{A}$ can be malicious in 5PC setting i.e., the corrupt parties can arbitrarily deviate from the protocol specification and can be both malicious and passive (honest but curious) in the 4PC setting. The computational security parameter is denoted by κ. A function $\mathsf{negl}(\kappa)$ is said to be *negligible* in κ if for every positive polynomial $p(\cdot)$, there exists an n_0 such that for all $n > n_0$, it holds that $\mathsf{negl}(n) < \frac{1}{p(n)}$. A *probability ensemble* $X = \{X(a, n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$ is an infinite sequence of random variables indexed by a and $n \in \mathbb{N}$. Two ensembles $X = \{X(a, n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$ and $Y = \{Y(a, n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $X \overset{c}{\approx} Y$, if for every PPT algorithm D, there exists a negligible function $\mathsf{negl}(.)$ such that for every $a \in \{0,1\}^*$ and $n \in \mathbb{N}$, $|\Pr[D(X(a, n)) = 1] - \Pr[D(Y(a, n)) = 1]| \leq \mathsf{negl}(n)$.

The security of our protocols is proven based on the standard real/ideal world paradigm i.e. it is examined by comparing the adversary's behaviour in a real execution to that of an ideal execution considered to be secure by definition (in presence of an incorruptible trusted third party (TTP)). In an ideal execution, each participating party sends its input to the TTP over a perfectly secure channel, the TTP computes the function using these inputs and sends respective output to each party. Informally, a protocol is said to be secure if an adversary's behaviour in the real protocol (where no TTP exists) can be simulated in the above described ideal computation. The formal definitions of the functionalities used to achieve the security notions of GOD, fairness and unanimous abort for a general polynomial function f, appear in Figs 2.1, 2.2, 2.3 respectively. These are motivated from [CL14, GLS15].

Functionality $\mathcal{F}_{\mathsf{god}}$

Each honest party P_i ($i \in [n]$) sends its input x_i to the functionality. Corrupted parties may send arbitrary inputs.

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

Output: Compute $y = f(x'_1, x'_2, x'_3, ..., x'_n)$ and send $(\mathtt{Output}, y)$ to party P_i for every $i \in [n]$.

Figure 2.1: Ideal Functionality $\mathcal{F}_{\mathsf{god}}$

Functionality $\mathcal{F}_{\mathsf{fair}}$

Each honest party P_i ($i \in [n]$) sends its input x_i to the functionality. Corrupted parties may send arbitrary inputs as instructed by the adversary. When sending the inputs to the functionality, the adversary is allowed to send a special `abort` command as well.

Input: On message (Input, x_i) from P_i, do the following: if $(\mathsf{Input}, *)$ message was received from P_i, then ignore. Otherwise record $x'_i = x_i$ internally. If x'_i is outside of the domain for P_i, consider $x'_i = \mathtt{abort}$.

Output: If there exists $i \in [n]$ such that $x'_i = \mathtt{abort}$, send $(\mathtt{Output}, \perp)$ to all the parties. Else, send $(\mathtt{Output}, y)$ to party P_i for every $i \in [n]$, where $y = f(x'_1, x'_2, x'_3, ..., x'_n)$.

Figure 2.2: Ideal Functionality $\mathcal{F}_{\mathsf{fair}}$

Functionality $\mathcal{F}_{\mathsf{uAbort}}$

Each honest party P_i ($i \in [n]$) sends its input x_i to the functionality. Corrupted parties may send arbitrary inputs as instructed by the adversary. When sending the inputs to the trusted party, the adversary is allowed to send a special `abort` command as well.

Input: On message (Input, x_i) from P_i, do the following: if $(\mathsf{Input}, *)$ message was received from

P_i, then ignore. Otherwise record $x'_i = x_i$ internally. If x'_i is outside of the domain for P_i, consider $x'_i = \texttt{abort}$.

Output to the adversary: If there exists $i \in [n]$ such that $x'_i = \texttt{abort}$, send $(\texttt{Output}, \perp)$ to all the parties. Else, send $(\texttt{Output}, y)$ to the adversary, where $y = f(x'_1, x'_2, x'_3, ..., x'_n)$.

Output to honest parties: Receive either continue or abort from the adversary. In case of continue, send y to all honest parties. In case of abort send $\perp$ to all honest parties.

Figure 2.3: Ideal Functionality $\mathcal{F}_{\mathsf{uAbort}}$

In the next section, we discuss the primitives that we use for our constructions.

2.2 Primitives

2.2.1 Garbling Scheme

We follow the circuit garbling approach to perform secure computation of a function formalized as a primitive by *Bellare et al* [BHR12]. A garbling scheme $\mathcal{G}$ is characterized by a tuple of four PPT algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ defined as follows:

- $\mathsf{Gb}(1^\kappa, C)$, transforms the circuit to be garbled C into a triplet $(\mathbf{C}, e, d)$ where $\mathbf{C}$ is the garbled circuit, e is input encoding information and d is output decoding information.
- $\mathsf{En}(e, x)$ maps the input x to garbled input $\mathbf{X}$ using input encoding information e.
- $\mathsf{Ev}(\mathbf{C}, \mathbf{X})$ produces garbled output $\mathbf{Y}$ by evaluating the garbled circuit $\mathbf{C}$ on garbled input $\mathbf{X}$.
- $\mathsf{De}(d, \mathbf{Y})$ decodes garbled output $\mathbf{Y}$ to clear output y using decoding information d.

We additionally use the property of a projective garbling scheme required in our protocols. A circuit $C : \{0,1\}^n \to \{0,1\}^m$ on garbling projectively generates encoding information, $e = (e_i^0, e_i^1)_{i\in[n]}$ and the encoded input corresponds to $\mathbf{X} = (e_i^{x_i})_{i\in[n]} = \mathsf{En}(x, e)$. We formally define the properties desired of our garbling scheme below.

2.2.1.1 Properties of Garbling Scheme

Definition 2.2.1. *A projective garbling scheme is one where while garbling a circuit $C : \{0,1\}^n \to \{0,1\}^m$, the e has the form $e = (e_i^0, e_i^1)_{i\in[n]}$, and $\boldsymbol{X}$ for $x = (x_i)_{i\in[n]}$ can be interpreted as $\mathbf{X} = \mathsf{En}(x, e) = (e_i^{x_i})_{i\in[n]}$.*

Definition 2.2.2. *A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is correct if for all input lengths $n \le \mathsf{poly}(\kappa)$, circuit $C : \{0,1\}^n \to \{0,1\}^m$ and inputs $x \in \{0,1\}^n$,*
$\mathsf{Pr}[\mathsf{De}(\mathsf{Ev}(\boldsymbol{C}, En(x, e)), d) \ne C(x) : (\boldsymbol{C}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)] \le \mathsf{negl}(\kappa)$

Definition 2.2.3. *A garbling scheme $\mathcal{G}$ is private if for all $n \leq \mathsf{poly}(\kappa)$, circuit $C : \{0,1\}^n \to \{0,1\}^m$, there exists a PPT simulator $\mathcal{S}_{\mathbf{priv}}$ such that for all $x \in \{0,1\}^n$, for all PPT adversary $\mathcal{A}$ the following distributions are computationally indistinguishable.*

- REAL(C, x): *run* $(\boldsymbol{C}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$ *and output* $(\boldsymbol{C}, \mathsf{En}(x, e), d)$

- IDEAL$(C, C(x))$: *output* $(\boldsymbol{C'}, \boldsymbol{X}, d') \leftarrow \mathcal{S}_{\mathbf{priv}}(1^\kappa, C, C(x))$

Definition 2.2.4. *A garbling scheme $\mathcal{G}$ is authentic if for all $n \leq \mathsf{poly}(\kappa)$, circuit $C : \{0,1\}^n \to \{0,1\}^m$, input $x \in \{0,1\}^n$ and for all PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\kappa)$.*

$$\mathsf{Pr}\begin{pmatrix} \hat{\boldsymbol{Y}} \neq \mathsf{Ev}(\boldsymbol{C}, \boldsymbol{X}) & : & \boldsymbol{X} = \mathsf{En}(x, e), (\boldsymbol{C}, e, d) \leftarrow \mathsf{Gb}(\kappa, C), \\ \wedge\, \mathsf{De}(\hat{\boldsymbol{Y}}, d) \neq \bot & & \hat{\boldsymbol{Y}} \leftarrow \mathcal{A}(\boldsymbol{C}, \boldsymbol{X}) \end{pmatrix}$$

2.2.2 Distributed Garbled Circuit

[BMR90, BLO16] In multiparty setting, it is necessary for all parties to participate in the construction of garbled circuit to prevent any coalition of corrupt parties from learning information about the value being computed. In the computation of distributed garbled circuit (DGC) with n parties, let $n-1$ $\{P_1, ..., P_{n-1}\}$ parties be the garblers and P_n be the evaluator. Each wire w is associated with mask $\lambda_w \in \{0,1\}$.P_i samples its mask share λ_w^i s.t $\oplus_{i \in [n-1]} \lambda_w^i = \lambda_w$. The technique of point and permute is used to hide the outputs of intermediate gates and λ_w acts as the permutation bit for each wire w. Every P_i chooses two keys $k_{w,0}^i, k_{w,1}^i = k_{w,0}^i \oplus \Delta^i$ per wire where Δ^i is the global offset of P_i. Each wire is thus defined with a set of $n-1$ keys for 0-label and $n-1$ keys for 1-label. The property of free-XOR allows the output key and mask of an XOR gate to be set equal to the XOR of the input keys and masks. Construction of AND gate ciphertexts requires interaction amongst the garblers and thus is realized by all garblers running a secure MPC protocol to compute the distributed garbled circuit.

2.2.3 Non-Interactive Commitment Schemes

A Non-Interactive Commitment Scheme (NICOM) is characterized by two PPT algorithms $(\mathsf{Com}, \mathsf{Open})$ for the purpose of commitment and opening phase defined as follows:

– Com outputs commitment c and corresponding opening information o, given a security parameter κ, a common public parameter pp, message x and random coins r.

– Open outputs the message x given κ, pp, a commitment c and corresponding opening information o.

The properties to be satisfied by a commitment scheme are:

– *Correctness:* For all values of public parameter pp, message $x \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, if $(c, o) \leftarrow \mathsf{Com}(x; r)$ then $\mathsf{Open}(c, o) = x$.

– *Hiding:* For all PPT adversaries $\mathcal{A}$, all values of pp, and all $x, x' \in \mathcal{M}$, the difference $|\mathsf{Pr}_{(c,o)\leftarrow\mathsf{Com}(x)}[\mathcal{A}(c) = 1] - \mathsf{Pr}_{(c,o)\leftarrow\mathsf{Com}(x')}[\mathcal{A}(c) = 1]|$ is negligible.

– *Binding*: A PPT adversary $\mathcal{A}$ outputs (c, o, o') such that $\mathsf{Open}(c, o) \neq \mathsf{Open}(c, o')$ and $\perp \notin \{\mathsf{Open}(c, o), \mathsf{Open}(c, o')\}$ with negligible probability over uniform choice of pp and random coins of $\mathcal{A}$.

We use instantiations based on injective one-way functions that ensure a strong binding even if the public parameter is arbitrarily chosen by adversary.

2.2.3.1 Instantiations

In the random oracle model, the commitment scheme is:

– $\mathsf{Com}(x; r)$ sets $c = H(x||r)$, $o = (x||r)$ where c, o refer to the commitment and opening respectively. The pp can be empty.

– $\mathsf{Open}(c, o = (x||r))$ returns x if $H(o) = c$ and $\perp$ otherwise.

For the purpose of all empirical results, the random oracle can be instantiated using a hash function. Alternatively, based on one-way permutation, we present an instantiation of NICOM(Com, Open) used theoretically in our protocols as: Let $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a one-way permutation and $h : \{0, 1\}^n \rightarrow \{0, 1\}$ be a hard-core predicate for f. Then the bit-commitment scheme for x is:

– $\mathsf{Com}(x, r)$ sets $c = (f(r), x \oplus h(r))$ where $r \in_R \{0, 1\}^n$ and $o = (x||r)$.

– $\mathsf{Open}(c, o = (x||r))$ returns x if $c = (f(r), x \oplus h(r))$, else $\perp$.

We provide bit and string based instantiations for NICOM($\mathsf{Com}, \mathsf{Open}$) [CGMV17] based on block ciphers that are secure in the ideal cipher model [Sha49, HKT11, Bla06] that are used in our AOT protocols for efficiency. The bit commitment scheme is as follows:

– $\mathsf{Com}(b, r)$ sets $c = F_k(r) \oplus r \oplus b^n$ where $b^n = ||_{i\in[n]}b$ and $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ is a random permutation parametrized by key k. Also, $o = (r||b)$.

– $\mathsf{Open}(c, o = (r||b))$ returns b if $c = F_k(r) \oplus r \oplus b^n$ and $\perp$ otherwise.

However, this bit commitment scheme is not secure for string commitments. Hence we describe the following secure instantiation:

- $\mathsf{Com}(m, r)$ sets $c = F_k(r) \oplus r \oplus F_k(m) \oplus m$ s.t $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ is a random permutation parametrized by key k and $o = (r||m)$.

- $\mathsf{Open}(c, o = (r||m))$ returns b if $c = F_k(r) \oplus r \oplus F_k(m) \oplus m$, else $\perp$.

2.2.4 Equivocal Non-Interactive Commitment Schemes

For our fair protocols, we need an equivocal NICOM (eNICOM). An eNICOM is defined with four PPT algorithms $(\mathsf{eCom}, \mathsf{eOpen}, \mathsf{eGen}, \mathsf{Equiv})$. $\mathsf{eCom}, \mathsf{eOpen}$ are defined as in NICOM and $\mathsf{eGen}, \mathsf{Equiv}$ are used to provide equivocation. Equiv enables a dummy commitment to be opened to any desired message with the help of a trapdoor output by eGen. These algorithms are defined as follows:

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

- $\mathsf{Equiv}(c, o', x, t)$ returns an o s.t $x \leftarrow \mathsf{eOpen}(\mathsf{epp}, c, o)$ when invoked on commitment c, its opening o', the desired message x (to which equivocation is required) and the trapdoor t.

An eNICOM should satisfy the following properties:

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

- *Hiding:* For all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$, all PPT adversaries $\mathcal{A}$ and all $x, x' \in \mathcal{M}$, the difference $|\mathsf{Pr}_{(c,o)\leftarrow\mathsf{eCom}(x)}[\mathcal{A}(c, o) = 1] - \mathsf{Pr}_{(c,o)\leftarrow\mathsf{eCom}(x),o\leftarrow\mathsf{Equiv}(c,x,t)}\mathcal{A}(c, o) = 1|$ is negligible.

- *Binding:* For all $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$, a PPT adversary $\mathcal{A}$ outputs $(c, o, o^{'})$ s.t $\mathsf{eOpen}(c, o) \neq \mathsf{eOpen}(c, o^{'})$ and $\perp \notin \{\mathsf{eOpen}(c, o), \mathsf{eOpen}(c, o^{'})\}$ with negligible probability.

2.2.4.1 Instantiations

We can use the equivocal bit commitment scheme of [CIO98] in the standard model, based on Naor's commitment scheme [Nao91] for bits. Let $\mathsf{G} : \{0,1\}^n \to \{0,1\}^{4n}$ be a pseudorandom generator. The commitment scheme for bit b used in the 5PC protocols is:

- $\mathsf{eGen}(1^\kappa)$ sets $(\mathsf{epp}, t_1, t_2, t_3, t_4) = ((\sigma, \mathsf{G}(r_1), \mathsf{G}(r_2), \mathsf{G}(r_3), \mathsf{G}(r_4)), r_1, r_2, r_3, r_4)$, where $\sigma = \mathsf{G}(r_1) \oplus \mathsf{G}(r_2) \oplus \mathsf{G}(r_3) \oplus \mathsf{G}(r_4)$. $t = ||_{i\in[4]} t_i$ is the trapdoor.

– $\mathsf{eCom}(x; r)$ sets $c = \mathsf{G}(s_1) \oplus \mathsf{G}(s_2)$ if $x = 0$, else $c = \mathsf{G}(s_1) \oplus \mathsf{G}(s_2) \oplus \sigma$ and sets $o = (x||r)$ where $r = s_1||s_2$.

– $\mathsf{eOpen}(c, o = (x||r))$ returns x if $c = \mathsf{G}(s_1) \oplus \mathsf{G}(s_2) \oplus x \cdot \sigma$ (where $(\cdot)$ denotes multiplication by a constant), else returns $\perp$.

– $\mathsf{Equiv}(c = \mathsf{G}(r_1) \oplus \mathsf{G}(r_2), \perp, x, (t_1, t_2, t_3, t_4))$ returns $o = (x||r)$ where $r = t_1||t_2$ if $x = 0$, else $r = t_3||t_4$. The entire trapdoor $t = (t_1, t_2, t_3, t_4)$ is required for equivocation.

For 4PC protocols, the eNICOM instantiation given above is modified as follows:

- $(\mathsf{epp}, t) \leftarrow \mathsf{eGen}(1^\kappa)$ where trapdoor $t = t_0||t_1$ and public parameter $\mathsf{epp} = (\sigma, G(t_0), G(t_1))$ s.t $\sigma = G(t_0) \oplus G(t_1)$.

- $\mathsf{eCom}(\mathsf{epp}, x)$ samples randomness r such that r and sets $c = G(r)$ if $x = 0$, else sets $c = G(r) \oplus \sigma$. It sets opening information $x = (x||r)$.

- $\mathsf{eOpen}(\mathsf{epp}, c, o = x||r)$ returns x if $c = G(r) \oplus x \cdot \sigma$ ($\cdot$ denotes multiplication by bit) , else returns $\perp$.

- $\mathsf{Equiv}(c = G(t_0), x, t = t_0||t_1)$ returns $o = x||t_0$ if $x = 0$, else returns $o = x||t_1$.

For empirical purposes, we rely on the random oracle based scheme presented before with the property of equivocation and is realized using a hash function.

2.2.5 Extractable Commitment Schemes

In this section, we consider a 3-round extractable commitment protocol (C, R). We now define extractable commitments taken verbatim from [PW09]:

Definition 2.2.5. *Let (C, R) be a statistically binding commitment scheme. We say that (C, R) is an extractable commitment scheme if there exists an expected polynomial-time probabilistic oracle machine (the extractor) E that given oracle access to any PPT cheating sender C^* outputs a pair (τ, σ^*) s.t*

- *(simulation) τ is identically distributed to the view of C^* at the end of interacting with an honest receiver in the commit phase*

- *(extraction) the probability that τ is accepting and $\sigma^* = \perp$ is negligible.*

- *(binding) if $\sigma^* \neq \perp$, then it is statistically impossible to open τ to any value other than σ^*.*

2.2.5.1 Instantiation

An instantiation of an extractable commitment $(\mathsf{ExtCom}, \mathsf{ExtOpen})$ appears in Fig 2.4. We refer to [PW09] for details of proof (implicit in [PRS02, Ros04]) that ExtCom is an extractable commitment scheme.

Protocol $\mathsf{ExtCom}, \mathsf{ExtOpen}$

Commitment phase ExtCom**:**

Let $\sigma \leftarrow \{0,1\}^m$ denote the input of S (committer / sender)

Round 1: S commits (using Ncom Com) to k pairs of strings $(v_1^0, v_1^1) \ldots (v_n^0, v_n^1)$ where $(v_i^0, v_i^1) = (\eta_i, \sigma \oplus \eta_i)$ and $\eta_1 \ldots \eta_k$ are random strings in $\{0,1\}^m$.

Round 2: R sends challenge $e = (e_1 \ldots e_k)$.

Round 3: S opens the commitments to $v_1^{e_1} \ldots v_k^{e_k}$. R checks if the openings are valid.

Decommitment Phase $\mathsf{ExtOpen}$**:**

- S sends σ and opens the commitments to all k pairs of strings.
- R checks that all the openings are valid and also that $\sigma = v_1^0 \oplus v_1^1 = \ldots v_k^0 \oplus v_k^1$.

Figure 2.4: Extractable Commitment Scheme

2.2.6 Secret Sharing Schemes

We use *additive sharing* and *replicated secret sharing* (RSS) [CDI05, ISN89] for our constructions. For a value x, its gth additive share is noted as x^g. We now recall RSS. Consider a secret x, of some finite field $\mathbb{F}$ to be shared among n parties s.t only $> t$ parties can reconstruct x. A *maximal unqualified* set is the set of t parties who together cannot reconstruct the secret. A dealer with secret x splits it into additive shares s.t each share corresponds to one *maximal unqualified* set $\mathcal{T}_l$, $l \in \{1, ..., \binom{n}{t}\}$. Formally, $x = \sum_{l \in [\binom{n}{t}]} x^l$. Each share x^l is associated with the unqualified set $\mathcal{T}_l$ (lexicographically wlog) and additive shares are random s.t they sum to x. Each party $P_i, i \in [n]$ gets all x^l for $P_i \notin \mathcal{T}_l$. This ensures that t parties alone of any $\mathcal{T}_l$ cannot retrieve the secret x. Specifically in our 5PC protocols, we use a 4-party RSS with $t = 2$ private against 2 corruptions where, each party gets 3 shares and each share is held by 3 parties including the dealer. Reconstruction is done by combining the shares held by any 3 parties. Given only shares of any two parties $\{P_i, P_j\}$, x remains private as x^l associated with $\mathcal{T}_l$ where $\mathcal{T}_l = \{P_i, P_j\}$ is missing from the view. Both additive secret sharing and RSS are instantiated over $\mathbb{F}_2$ for our protocols.

2.2.7 Collision Resistant Hash

[RS04] Consider a hash function family $\mathsf{H} = \mathcal{K} \times \mathcal{L} \rightarrow \mathcal{Y}$. The hash function H is said to be collision resistant if for all probabilistic polynomial-time adversaries $\mathcal{A}$, given the description of H_k where $k \in_R \mathcal{K}$, there exists a negligible function $\mathsf{negl}()$ such that $\Pr[(x_1, x_2) \leftarrow \mathcal{A}(k) : (x_1 \neq x_2) \wedge \mathsf{H}_k(x_1) = \mathsf{H}_k(x_2)] \leq \mathsf{negl}(\kappa)$, where $m = \mathsf{poly}(\kappa)$ and $x_1, x_2 \in_R \{0,1\}^m$.

2.2.8 Oblivious Transfer

Oblivious transfer (OT) [EGL85] is one of the most fundamental building blocks in secure computation. OT is a protocol between two parties: a sender and a receiver. Informally, OT protocol is a type of protocol in which a sender transfers one of potentially many pieces of information to a receiver, but remains *oblivious* as to what piece (if any) has been transferred. For oblivious transfer, we denote the sender by S and the receiver by R. In a 1-out-of-2 OT on ℓ bit strings, S holds two inputs x_0, x_1, each from $\{0,1\}^\ell$ and R holds a *choice bit* b. The output to R is x_b and R remains unaware about x_{1-b}. The sender S remains oblivious as to which of x_0, x_1 has been received by R. The formal functionality is presented in Fig 2.5.

Functionality $\mathcal{F}_{\mathsf{OT}}$

Choose: On input (rec, σ) from R where $\sigma \in \{0,1\}$; if no message of the form (rec, σ) has been recorded in memory, store (rec, σ) and send rec to S.

Transfer: On input $(\mathsf{sen}, (x_0, x_1))$ from S with $x_0, x_1 \in \{0,1\}^n$, if no message of the form $(\mathsf{sen}, (x_0, x_1))$ is recorded and a message of the form (rec, σ) is stored, send $(\mathsf{sent}, x_\sigma)$ to R and sent to S.

Figure 2.5: Ideal Functionality for OT $\mathcal{F}_{\mathsf{OT}}$.

Chapter 3

Distributed Garbling and More

At the heart of our 5PC and 4PC lie a 4-party (4DG) and 3-party (3DG) distributed garbling (DG) respectively and a matching evaluation protocol tolerating arbitrary *semi-honest* corruptions. For better understanding, we first concretely describe the 4-party garbling scheme and the matching evaluation protocol. Then, we provide details to trivially scale down the 4DG to 3DG.

Garbling is done distributively amongst the garblers $\{P_1, P_2, P_3, P_4\}$ and P_5 enacts the sole evaluator. Our distributed garbling scheme is a direct simplification of the state-of-the-art actively-secure distributed garbling scheme of [WRK17]. The semi-honest scheme when combined with party-emulation idea of [CGMV17], achieves malicious security against 2 corruptions. Specifically, the role of each garbler in the underlying semi-honest 4DG scheme is *also* enacted by two other fellow garblers. This emulation is achieved via a unique seed distribution (SD) technique that ensures that the seed of a garbler is consistent with two other garblers and all the needed randomness for 4DG is generated from the seed. This helps to detect any wrong-doing by at most two garblers. Interestingly, the seed distribution can further be leveraged to replace the computationally-heavy public-key primitive Oblivious Transfer (OT) in [WRK17] with an inexpensive symmetric-key based alternative called *attested* OT [CGMV17]. While all our protocols for 5PC can be realized with any underlying passively-secure garbling scheme when used with SD and attested OT, we choose the current construction for efficiency. We start with the building blocks of 5PC.

3.1 Building Blocks for 5PC

3.1.1 Seed Distribution

The starting point of our 5PC protocols is a semi-honest distributed garbling with $\{P_1, P_2, P_3, P_4\}$ as garblers and P_5 as evaluator. The final distributed garbled circuit (DGC) is denoted as $GC = GC^1||GC^2||GC^3||GC^4$. In distributed garbling, all randomness required by a garbler P_i is generated using a random seed s_i. The SD technique involves distributing the seeds among 4 garblers s.t the seed s_i generated by P_i is held by two other garblers and no single garbler has the knowledge of all 4 seeds. Consequently, any data computed based on s_i is done identically by 3 parties who own s_i and thus, can be compared for correctness. With at least one honest party in this team of 3 parties, any wrong-doing by at most two parties is detected. The SD functionality $\mathcal{F}_\mathsf{S}$ is depicted in Fig 3.1 and is realized differently in each of our protocols based on the required security guarantee (fairness, unanimous abort or GOD). We use $\mathcal{S}_g$ to denote the set of indices of parties who hold s_g as well as the set of indices of the seeds held by party P_g. Note that both these sets are identical– for instance, $\mathcal{S}_1 = \{1, 3, 4\}$ indicates that parties P_1, P_3, P_4 hold s_1. $\mathcal{S}_1$ also indicates that P_1 holds $\mathsf{s}_1, \mathsf{s}_3, \mathsf{s}_4$. Thus, the fragment GC^1 (analogously GC^2, GC^3 and GC^4) are constructed by three parties P_1, P_3, P_4 who hold seed s_1.

Functionality $\mathcal{F}_\mathsf{S}$

Let $\mathcal{S}_i, i \in [4]$ be $\mathcal{S}_1 = \{1, 3, 4\}$, $\mathcal{S}_2 = \{2, 3, 4\}$, $\mathcal{S}_3 = \{1, 2, 3\}$, $\mathcal{S}_4 = \{1, 2, 4\}$. Let $\mathcal{H} \subset \mathcal{P}, \mathcal{C} \subset \mathcal{P}$ be the set of indices of Honest and Corrupt parties respectively. Each honest party $P_g, g \in \mathcal{H})$ sends its input $(\mathsf{Input}, *)$ to the functionality. Corrupted parties $P_j, j \in \mathcal{C}$ may send the trusted party $(\mathsf{Input}, \mathsf{s}_j/\perp)$ as instructed by the adversary.

On message $(\mathsf{Input}, *)$ from garbler $P_g, g \in \mathcal{H}$ and $(\mathsf{Input}, \{\mathsf{s}_j/\perp\}_{j\in\mathcal{C}})$ from adversary, sample s_g on behalf of every honest P_g. Send each seed $\mathsf{s}_i, i \in [4]$ (or $\perp$ as given by adversary) to each party in $\mathcal{S}_i$.

Figure 3.1: Ideal Functionality $\mathcal{F}_\mathsf{S}$

3.1.2 Attested Oblivious Transfer

The Attested Oblivious Transfer (AOT) protocol [CGMV17] can be viewed as an OT between a sender and a receiver with additional help from two other parties called “attesters”. These “attesters” aid in ensuring correctness of the OT protocol by attesting inputs of the sender and the receiver, thus tolerating 2 active corruptions. AOT functionality is recalled in Fig 3.2.

Functionality $\mathcal{F}_{\mathsf{4AOT}}$

P_s acts as sender, P_r acts as receiver and P_{a_1}, P_{a_2} act as attesters.

- On input message ($\mathtt{Sen}$, m_0, m_1) from P_s, record (m_0, m_1) and send ($\mathtt{Sen}$, m_0, m_1) to P_{a_1} and P_{a_2} and $\mathtt{Sen}$ to the adversary.
- On input message ($\mathtt{Rec}$, b) from P_r, where $b \in \{0, 1\}$, record b and send ($\mathtt{Rec}$, b) to P_{a_1} and P_{a_2} and $\mathtt{Rec}$ to the adversary.
- On input message ($\mathtt{A}$, m_0^j, m_1^j, b^j) from $P_{a_j}, j \in [2]$, if ($\mathtt{Sen}$, sid, $*$, $*$) and ($\mathtt{Rec}$, $*$) have been recorded, ignore this message; otherwise, record ($m_0^{a_j}$, $m_1^{a_j}$, b^{a_j}) and send $\mathtt{A}$ to the adversary.
- On input message $\mathtt{Output}$ from the adversary, if $(m_0, m_1, b) \neq (m_0^{a_1}, m_1^{a_1}, b^{a_1})$ or $(m_0, m_1, b) \neq (m_0^{a_2}, m_1^{a_2}, b_{a_2})$, send $(\mathtt{Output}, \perp)$ to P_r; else send $(\mathtt{Output}, m_b)$ to P_r.
- On input message $\mathtt{abort}$ from the adversary, send $(\mathtt{Output}, \perp)$ to P_r.

Figure 3.2: Ideal Functionality $\mathcal{F}_{\mathsf{4AOT}}(P_s, P_r, \{P_{a_1}, P_{a_2}\})$ for 4DG

For the attested OT functionality $\mathcal{F}_{\mathsf{4AOT}}$ defined in Fig 3.2, we now provide a standalone instantiation. The sender of the AOT, P_s having inputs m_0, m_1 samples random $r_0, r_1 \leftarrow \{0,1\}^\kappa$ and generates the commitments: $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$, $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_s sends (m_0, r_0, m_1, r_1) to the attesters and $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$ to the receiver. The receiver P_r sends the choice bit b to the attesters. The attesters exchange the copy of messages received from P_s, P_r amongst themselves to verify correctness. If verified, they use (m_0, r_0, m_1, r_1) to compute the commitments $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$ and send the same to the receiver. One of the attesters, say P_{a_1} also sends the opening corresponding to c_b to P_r. If the verification fails, the attesters send $\perp$ to P_r. The receiver P_r then checks if all the copies of commitments received are the same. If not, aborts. Else, P_r uses the opening of c_b to obtain m_b.

3.1.3 The semi-honest 4DG and Evaluation

A distributed garbled circuit (DGC) is prepared together by all garblers in a distributed manner. Each wire w in our 4DG scheme is associated with a mask bit $\lambda_w \in \{0, 1\}$ and each garbler P_g holds a share λ_w^g s.t $\lambda_w = \oplus_{g \in [4]} \lambda_w^g$. Each P_g samples two keys $k_{w,0}^g$, $k_{w,1}^g = k_{w,0}^g \oplus \Delta^g$ for each wire w, with global offset Δ^g. Thus, each super-key of a wire has 4 keys contributed by 4 garblers.

Definition 3.1.1. *A* super-key *of a wire is a set of* 4 *keys, each contributed by one garbler i.e.,* $\{k_{w,0}^g\}_{g \in [4]}$ *indicates the 0-super-key on wire* w *and* $\{k_{w,1}^g\}_{g \in [4]}$ *indicates the 1-super-key on* w.

Free-XOR is enabled by setting the mask and keys for the output wire of an XOR gate as the XOR of masks and keys of its input wires. A garbled AND gate, on the other hand, comprises of 4 super-ciphertexts (super-CT), one for each row of truth table. A super-CT is made up of 4 CTs, each of which is contributed by one garbler. Each CT hides a share of a super-key on the output wire such that during evaluation, 4 decrypted messages of a super-CT together would give the desired super-key on the output wire. In order to hide the actual output of intermediate gates from an evaluator, we enable *point and permute*. The mask bit λ_w acts as the permutation bit for wire w. Thus, for an AND gate with input wires u, v, output wire w and their corresponding masks λ_u, λ_v, λ_w, if x_u, x_v denote the actual values on wires u, v respectively, then the evaluator sees super-keys k^g_{u,b_u}, k^g_{v,b_v} where b_u, b_v defined as $(b_u = x_u \oplus \lambda_u), (b_v = x_v \oplus \lambda_v)$ denote the blinded bits. The evaluator then decrypts the super-CT positioned at row (b_u, b_v) and obtains the output super-key $\{k^g_{w,0} \oplus \Delta^g(x_u x_v \oplus \lambda_w)\}_{g\in[4]}$ that corresponds to the blinded (masked) bit $x_u x_v \oplus \lambda_w$ on wire w.

Definition 3.1.2. *A* blinded *or* masked *bit of a bit x_w on a wire w is the XOR of x_w with mask bit λ_w on wire w i.e. $b_w = x_w \oplus \lambda_w$.*

Interpreting row (b_u, b_v) as $\gamma = 2b_u + b_v + 1$ and recasting the above, we see that the super-CT at row γ for $\gamma \in [4]$ encrypts the super-key $\{k^g_{w,0} \oplus \Delta^g((b_u \oplus \lambda_u)(b_v \oplus \lambda_v) \oplus \lambda_w)\}_{g\in[4]}$. In 4DG, the super-CTs as above for an AND gate are prepared distributedly amongst the garblers, using the additive shares of the mask bits and keys held by each garbler corresponding to the input and output wires of the gate. We achieve this in a two-step process. First, we generate the additive sharing of each key belonging to the super-key to be encrypted in each row. Second, for each row, a garbler encrypts the *additive shares* it holds of each key of the corresponding super-key (obtained in the first step) in the CT that it contributes for the super-CT of that row. A CT for row γ has the format of one-time pad where the pad is calculated using a double-keyed PRF with keys corresponding to row γ.

Definition 3.1.3. *A* super-ciphertext *for a given row γ ($\gamma = 2b_u + b_v + 1$), of an AND gate with input wires u, v, output wire w, is a set of 4 CTs, $\{c^g_\gamma\}_{g\in[4]}$, where P_g contributes c^g_γ that encrypts its additive share of each key in $\{k^g_{w,0} \oplus \Delta^g((b_u \oplus \lambda_u)(b_v \oplus \lambda_v) \oplus \lambda_w)\}_{g\in[4]}$.*

To compute the additive sharing of super-key $\{k^g_{w,0} \oplus \Delta^g((b_u \oplus \lambda_u)(b_v \oplus \lambda_v) \oplus \lambda_w)\}_{g\in[4]}$ for all rows (i.e. all possibilities of (b_u, b_v)), we compute the additive sharing of the following in sequence, starting with the additive shares of $\lambda_u, \lambda_v, \lambda_w$: (A) $\lambda_u\lambda_v$ (for row 1 i.e. $\gamma = 1$ and $b_u = b_v = 0$), $\lambda_u\overline{\lambda_v}$ (for $\gamma = 2$ and $b_u = 0$, $b_v = 1$), $\overline{\lambda_u}\lambda_v$ (for $\gamma = 3$ and $b_u = 1$, $b_v = 0$) and $\overline{\lambda_u}\,\overline{\lambda_v}$ (for $\gamma = 4$ and $b_u = 1$, $b_v = 1$); (B) $\lambda_1 = \lambda_u\lambda_v \oplus \lambda_w, \lambda_2 = \lambda_u\overline{\lambda_v} \oplus \lambda_w, \lambda_3 = \overline{\lambda_u}\lambda_v \oplus \lambda_w, \lambda_4 = \overline{\lambda_u}\,\overline{\lambda_v} \oplus \lambda_w$; (C) $\Delta^g\lambda_\gamma$ for all $g, \gamma \in [4]$ and lastly (D) $k^g_{w,0} \oplus \Delta^g\lambda_\gamma$ for all $g, \gamma \in [4]$. (B) and (D) require linear operations, thus can be done locally by each garbler. However, for (A) and (C), additive sharing of a product needs to be computed which requires interaction among garblers.

This is done via OTs, which we explain below. Also, in (A), it is known how to tweak shares of $\lambda_u\lambda_v$ locally to get the shares of remaining products [BLO16], thus computing the sharing of $\lambda_u\lambda_v$ alone suffices. We now explain how the additive sharing of 1) $\lambda_u\lambda_v$ and 2) $\Delta^g\lambda_\gamma$ for any $\gamma \in [4]$ is computed.

To compute 1), each garbler P_g locally computes $\lambda_u^g\lambda_v^g$. In addition, each pair of parties $P_g, P_{g'}$ for $g \neq g'$ run an OT with P_g as sender, holding $(r, r \oplus \lambda_u^g)$ and $P_{g'}$ as receiver, holding $\lambda_v^{g'}$ to generate 2-out-of-2 additive sharing of $\lambda_u^g\lambda_v^{g'}$. P_g outputs its share as r denoted by $[\lambda_u^g\lambda_v^{g'}]_S$ and $P_{g'}$ outputs its share as the OT output $r \oplus \lambda_u^g\lambda_v^{g'}$ denoted by $[\lambda_u^g\lambda_v^{g'}]_R$ (We use $[\cdot]_S, [\cdot]_R$ to denote the shares of sender and receiver of OT respectively). Each garbler P_g now computes its share, λ_{uv}^g, of the product $\lambda_{uv} = \lambda_u\lambda_v$ as the sum of its local product $\lambda_u^g\lambda_v^g$ and the shares obtained from OTs either as a sender or as a receiver i.e., $\lambda_{uv}^g = \lambda_u^g\lambda_v^g \oplus (\oplus_{g \neq g'}[\lambda_u^g\lambda_v^{g'}]_S) \oplus (\oplus_{g \neq g'}[\lambda_u^{g'}\lambda_v^g]_R)$. Next, to compute 2), where Δ^g belongs to P_g and $\Delta^g\lambda_\gamma = \Delta^g(\lambda_\gamma^1 \oplus \lambda_\gamma^2 \oplus \lambda_\gamma^3 \oplus \lambda_\gamma^4)$, each garbler P_g first locally computes $\Delta^g\lambda_\gamma^g$ and then for each cross-term $\Delta^g\lambda_\gamma^{g'}, g \neq g'$, P_g acts as a sender with each $P_{g'}$ as receiver in an OT to get their respective shares $[\Delta^g\lambda_\gamma^{g'}]_S$ and $[\Delta^g\lambda_\gamma^{g'}]_R$. Finally, the share of P_g for the product $\Delta^g\lambda_\gamma$ is set to the following sum: $\Delta^g\lambda_\gamma^g \oplus (\oplus_{g' \neq g}[\Delta^g\lambda_\gamma^{g'}]_S)$, while the share of each $P_{g'}$ is set to $[\Delta^g\lambda_\gamma^{g'}]_R$. We now present the functionality $\mathcal{F}_{\mathsf{GC}}$ (Fig 3.3). Partitioning the set of all super-CTs into its 4 constituent CTs, we can view the GC as $GC^1 \mid\mid GC^2 \mid\mid GC^3 \mid\mid GC^4$ where gth partition is contributed by garbler P_g.

Functionality $\mathcal{F}_{\mathsf{GC}}$

Let C be the circuit, κ, the security parameter and F, a double-keyed PRF [BLO16]. Each garbler P_g prepares the private input set ISet_g consisting of:

- An offset string $\Delta^g \in \{0,1\}^\kappa$.
- A share $\lambda_w^g \in \{0,1\}$ of the masking bit for each wire w, barring the output wire of XOR gates.
- Keys $k_{w,0}^g, k_{w,1}^g \in \{0,1\}^\kappa$ for every wire w s.t $k_{w,1}^g = k_{w,0}^g \oplus \Delta^g$,except the output wire of XOR gates.

Input: On receiving message $(\mathsf{Input}, \mathsf{ISet}_g)$ from each garbler $P_g, g \in [4]$, compute super-keys and mask bits for all wires (those for XOR output wires are computed as per free-XOR). For every AND gate with input wires u, v; output wire w, the g^{th} CT in the γ^{th} super-CT for $g, \gamma \in [4]$ is computed as follows. For $a, b \in \{0,1\}$, let $\gamma = 2a + b + 1$, $\lambda_1 = \lambda_u\lambda_v \oplus \lambda_w, \lambda_2 = \lambda_u\overline{\lambda_v} \oplus \lambda_w, \lambda_3 = \overline{\lambda_u}\lambda_v \oplus \lambda_w, \lambda_4 = \overline{\lambda_u}\,\overline{\lambda_v} \oplus \lambda_w$, $\lambda_\gamma = \oplus_{g\in[4]}\lambda_\gamma^g$ and $[\Delta^{g'}\lambda_\gamma]_g$ denote the g^{th} additive share of $\Delta^{g'}\lambda_\gamma$, $g' \in [4]$.

$$c^g_\gamma = \underbrace{\mathsf{F}_{k^g_{u,a}, k^g_{v,b}}(w||g)}_{\text{Pad}} \oplus (\underbrace{\lambda^g_\gamma}_{\substack{\text{share of}\\\text{blinded}\\\text{output}}} || \underbrace{\{[\Delta^{g'}\lambda_\gamma]_g\}_{g' \neq g}}_{\substack{P_g\text{'s share of the}\\\text{output key of } P_{g'}}} || \underbrace{k^g_{w,0} \oplus [\Delta^g \lambda_\gamma]_g}_{\substack{P_g\text{'s share of the}\\\text{output key of } P_g}})$$

Output: On receiving Output from parties, send gth partition $GC^g = \{\{c^g_\gamma\}_{\gamma\in[4]\forall \text{ AND gates}}\}|| \{\{\mathsf{H}(k^g_{w,0}), \mathsf{H}(k^g_{w,1})\}_{\forall \text{ output wires w}}\}$ to P_g where H is the collision resistant hash (Section 2.2).

Figure 3.3: Ideal Functionality $\mathcal{F}_{\mathsf{GC}}$

Evaluation of the DGC Starting with the masked bits of all inputs and corresponding super-keys, P_5 evaluates a DGC in topological order, with XOR gates evaluated using free-XOR. For an AND gate with input wires u,v, P_5, given input super-keys $\{(k^g_{u,b_u}, k^g_{v,b_v})\}_{g\in[4]}$ and blinded input bits b_u, b_v, decrypts (b_u, b_v)th row's super-CT to obtain the super-key corresponding to blinded output bit $x_u x_v \oplus \lambda_w$ and the blinded output bit itself. The blinded bits for output wires give clear output when XORed with their respective masks.

3.1.3.1 4DG with AOT and Seed distribution

As iterated before, we assume that all the randomness required by a party P_g for 4DG is generated using a random seed s_g. The SD then enables a party-emulation technique where the seed s_g of P_g is available to exactly two other garblers in $\mathcal{S}_g$ who can now emulate the role of P_g. Thus, each partition of GC, GC^g is generated by 3 garblers holding s_g, offering security against at most two corrupt garblers. This also preserves input privacy as: (i) when two garblers are corrupt (and together hold all seeds), the evaluator is surely honest and protects the privacy of inputs; (ii) when a garbler and the evaluator are corrupt, one seed remains hidden, assuring input privacy. The SD results brings a prime gain in the underlying semi-honest 4DG– *replacing standard OTs with 1-round AOTs:* The standard OTs used to compute each cross-term $\lambda^g_u \lambda^{g'}_v$, $\Delta^g \lambda^{g'}_\gamma$ $(g \neq g')$ in the additive-sharing of $\lambda_u\lambda_v$,$\Delta^g\lambda_\gamma$ respectively, are replaced with AOTs. The SD further enables each AOT to be run s.t the attesters hold both seeds that the sender and receiver mutually-exclusively hold. This implies that the attesters are aware of the inputs of both sender and receiver at the onset, thus leading to a *one-round* instantiation of AOT. To elaborate, for instance in $\mathcal{F}_{\mathsf{4AOT}}$ (Fig 3.2), when $P_s = P_1, P_r = P_2$, the attesters are P_3, P_4 and the inputs of sender are derived from the seed s_1, while the input of the receiver is derived from seed s_2 (both seeds are with P_3, P_4). Thus, P_s, now sends $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$ to P_r and the attesters send $\mathsf{H}(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$ to P_r. Also, P_{a_1} sends opening corresponding to the commitment c_b. All these steps can be done parallelly in only one round and hence AOT in our garbling scheme needs only one round. P_r then computes the output as in the standalone description. This process is formally depicted in Fig 3.4.

Functionality Π_{4AOT}

P_s, P_r denote the sender and receiver respectively. P_{a_1}, P_{a_2} are attesters. All are distinct parties.

Inputs: P_s holds m_0, m_1, P_r holds choice bit b.

Output P_r outputs $m_b/\perp$.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Section 2.2).

- P_s samples pp and random $r_0, r_1 \leftarrow \{0,1\}^\kappa$ (derived from s_i, $i \in \mathcal{S}_s \setminus \mathcal{S}_r$) and computes $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$, $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_s sends $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$ to P_r. P_{a_1}, P_{a_2} who know (r_0, r_1) (since they know s_i) also compute $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$, $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$ and each send $\mathsf{H}((\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1))$ to P_r[a].
- P_r has b (derived using $\mathsf{s}_j, j \in \mathcal{S}_r \setminus \mathcal{S}_s$) which is known to P_{a_1}, P_{a_2} (since they know s_j). P_{a_1} (wlog) sends o_b to P_r.

(Local Computation by P_r): If the commitment sent by P_s and the hash values sent by P_{a_1}, P_{a_2} do not match, then P_r outputs $\perp$. Else, output $m_b = \mathsf{Open}(\mathsf{c}_b, \mathsf{o}_b)$.

[a] The exact realization of the functionality $\mathcal{F}_{\mathsf{4AOT}}$ involves P_s and P_r sending (r_0, m_0, r_1, m_1) and b respectively to P_{a_1} and P_{a_2} who in turn exchange their copies received from P_s, P_r for correctness.

Figure 3.4: Protocol $\Pi_{\mathsf{4AOT}}(P_s, P_r, \{P_{a_1}, P_{a_2}\})$ for 4DG realizing $\mathcal{F}_{\mathsf{4AOT}}$

Note that the party-emulation technique does not increase the number of OTs required to three times the underlying semi-honest 4DG but instead keeps it the same, since SD ensures that, for each garbler P_i, OTs are needed in the computation of every $\lambda_u^g \lambda_v^{g'}$, $\Delta^g \lambda_\gamma^{g'}$ $(g \neq g')$ only when one of g, g' is not in $\mathcal{S}_i$.

For clarity, below we demonstrate, how a particular product share λ_{uv}^1 (of $\lambda_u \lambda_v$) is computed by parties in $\mathcal{S}_1$ $(\{P_1, P_3, P_4\})$, utilizing AOT and SD. The share λ_{uv}^1 consists of summands as listed in the first column of the table below. We explain how P_1 computes each summand. Except $\lambda_u^1 \lambda_v^1$, the remaining summands correspond to cross-terms that P_1 originally obtained via OT either as sender or receiver. Now, all summands that correspond to P_1 enacting a sender $(\lambda_u^1 \lambda_v^g, g \neq 1)$ can be sampled from s_1, as the sender's share is a random bit. For the summands where P_1 enacts receiver $(\lambda_u^g \lambda_v^1, g \neq 1)$, AOT is needed only for the summand, $\lambda_u^2 \lambda_v^1$ that involves s_2 which P_1 does not own, while for other terms, P_1 can locally compute its share with the knowledge of both seeds. As for the AOT, P_1 acts as receiver with seed s_1, P_2 acts as sender with seed s_2, and $\{P_3, P_4\}$ act as attesters with $\{\mathsf{s}_1, \mathsf{s}_2\}$. Similarly, $\{P_3, P_4\}$ can compute the summands of λ^1_{uv} as indicated in the table.

Summand	$P_1 : (\mathsf{s}_1, \mathsf{s}_3, \mathsf{s}_4)$	$P_3 : (\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3)$	$P_4 : (\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_4)$
$\lambda^1_u \lambda^1_v$	local	local	local
$[\lambda^1_u \lambda^2_v]_S$ $[\lambda^1_u \lambda^3_v]_S$, $[\lambda^1_u \lambda^4_v]_S$	local	local	local
$[\lambda^2_u \lambda^1_v]_R$	$\mathcal{F}_{\mathsf{4AOT}}(P_2, P_1, \{P_3, P_4\})$	local	local
$[\lambda^3_u \lambda^1_v]_R$	local	local	$\mathcal{F}_{\mathsf{4AOT}}(P_2, P_4, \{P_1, P_3\})$
$[\lambda^4_u \lambda^1_v]_R$	local	$\mathcal{F}_{\mathsf{4AOT}}(P_2, P_3, \{P_1, P_4\})$	local

Our final garbling and evaluation protocols appear in Figs 3.5-3.6. Our 4DG scheme with the use of standard OTs [EGL85] can be scaled in a straightforward way to arbitrary n-parties tolerating at-most n-1 corruptions by setting each of n-1 parties to enact the role of a garbler and the remaining party to enact the role of an evaluator. However, with the use of AOTs, our 4DG scheme can be scaled in a straightforward way to arbitrary n-parties but tolerating at-most $\sqrt{n}$ corruptions. For completeness, we describe the semi-honest scheme when scaled to 3DG scheme in Section 3.2.3.

Protocol $\mathsf{Garble}_4()$

Common Inputs: Circuit C that computes f.

Primitives and Notation: A double-keyed PRF F [BLO16]. $\mathcal{S}_g$ denotes the indices of parties who hold s_g as well as the indices of seeds held by P_g.

Output: Each party $P_g, g \in [4]$ outputs $GC^j, j \in \mathcal{S}_g$ or $\perp$.

Sampling Phase: Each $P_g, g \in [4]$ samples Δ^j from $\mathsf{s}_j, j \in \mathcal{S}_g$. Also, the following is done for each wire w in C corresponding to seed s_j:

– If w is not an output wire of XOR gate, sample λ^j_w and $k^j_{w,0}$ from s_j. Set $k^j_{w,1} = k^j_{w,0} \oplus \Delta^j$.

– If w is an output wire of XOR gate with input wires u, v, set $\lambda^j_w = \lambda^j_u \oplus \lambda^j_v$, $k^j_{w,0} = k^j_{u,0} \oplus k^j_{v,0}$ and $k^j_{w,1} = k^j_{w,0} \oplus \Delta^j$.

The mask and super-key pair for a wire w is defined as $\lambda_w = \oplus_{g\in[4]} \lambda^g_w$ and $\left(\{k^g_{w,0}\}_{g\in[4]}, \{k^g_{w,1}\}_{g\in[4]}\right)$.

Run in parallel for every AND gate in C with input wires u, v and output wire w:

R1: Product Phase I: Define $\lambda_{uv} = \lambda_u \lambda_v = (\oplus_{g\in[4]} \lambda^g_u)(\oplus_{g\in[4]} \lambda^g_v)$. Likewise define $\lambda_{u\overline{v}}, \lambda_{\overline{u}v}, \lambda_{\overline{u}\,\overline{v}}$ that can be derived from shares of λ_{uv}. Each garbler P_g computes λ^j_{uv} of λ_{uv} for every $j \in \mathcal{S}_g$ as below:

– locally compute $\lambda_u^j \lambda_v^j$. For each $k \neq j$, sample $[\lambda_u^j \lambda_v^k]_S$ from seed s_j.
– for every $k \in \mathcal{S}_g$, locally compute $[\lambda_u^k \lambda_v^j]_R = [\lambda_u^k \lambda_v^j]_S \oplus \lambda_u^k \lambda_v^j$ with the knowledge of s_j and s_k.
– for every $k \notin \mathcal{S}_g$, obtain $[\lambda_u^k \lambda_v^g]_R$ from $\mathcal{F}_{\mathsf{4AOT}}$ acting as receiver with input λ_v^g and P_k as the sender with inputs $([\lambda_u^k \lambda_v^g]_S, [\lambda_u^k \lambda_v^g]_S \oplus \lambda_u^k)$ derived from s_k.
– for each $k \notin \mathcal{S}_g$, $j \neq g$, obtain $[\lambda_u^k \lambda_v^j]_R$ from $\mathcal{F}_{\mathsf{4AOT}}$ acting as a receiver with input λ_v^j, and sender $P_s, s = [4] \setminus \{g, j, k\}$ with inputs $([\lambda_u^k \lambda_v^j]_S, [\lambda_u^k \lambda_v^j]_S \oplus \lambda_u^k)$ derived from s_k.
– compute $\lambda_{uv}^j = \lambda_u^j \lambda_v^j \oplus (\oplus_{i \neq j} [\lambda_u^j \lambda_v^i]_S) \oplus (\oplus_{i \neq j} [\lambda_u^i \lambda_v^j]_R)$.

Define $\lambda_1 = \lambda_u \lambda_v \oplus \lambda_w, \lambda_2 = \lambda_u \overline{\lambda_v} \oplus \lambda_w, \lambda_3 = \overline{\lambda_u} \lambda_v \oplus \lambda_w, \lambda_4 = \overline{\lambda_u}\, \overline{\lambda_v} \oplus \lambda_w$. Every P_g computes jth share λ_1^j of λ_1 for all $j \in \mathcal{S}_g$ as $\lambda_{uv}^j \oplus \lambda_w^j$. Similarly, it computes the shares for $\lambda_2, \lambda_3, \lambda_4$.

R2: Product Phase II: P_g computes share $[\Delta^j \lambda_\gamma]_j$ (jth additive share) of $\Delta^j \lambda_\gamma$ for every $\gamma \in [4]$ and $j \in \mathcal{S}_g$ as follows:
– locally compute $\Delta^j \lambda_\gamma^j$. For every $k \neq j$, sample $[\Delta^j \lambda_\gamma^k]_S$ from s_j.
– compute $[\Delta^j \lambda_\gamma]_j = \Delta^j \lambda_\gamma^j \oplus_{k \neq j} [\Delta^j \lambda_\gamma^k]_S$.

P_g computes $[\Delta^k \lambda_\gamma]_j$ of $\Delta^k \lambda_\gamma$ for each $k \neq j$, $\gamma \in [4], j \in \mathcal{S}_g$ as:
∘ For every $k \in \mathcal{S}_g$, compute $[\Delta^k \lambda_\gamma]_j = [\Delta^k \lambda_\gamma^j]_R$ locally from the knowledge of s_j and s_k.
∘ For $k \notin \mathcal{S}_g$, $j = g$, obtain $[\Delta^k \lambda_\gamma^g]_R$ from $\mathcal{F}_{\mathsf{4AOT}}$ acting as receiver with input λ_γ^g and with P_k as sender whose inputs are $[\Delta^k \lambda_\gamma^g]_S$ and $[\Delta^k \lambda_\gamma^g]_S \oplus \Delta^k$ derived from s_k. Set $[\Delta^k \lambda_\gamma]_j = [\Delta^k \lambda_\gamma^j]_R$.
∘ For $k \notin \mathcal{S}_g$, $j \neq g$, obtain $[\Delta^k \lambda_\gamma^j]_R$ from $\mathcal{F}_{\mathsf{4AOT}}$ acting as receiver with input λ_γ^j and $P_s, s = [4] \setminus \{g, j, k\}$ as sender with inputs $[\Delta^k \lambda_\gamma^j]_S$, $[\Delta^k \lambda_\gamma^j]_S \oplus \Delta^k$ (from s_k). Set $[\Delta^k \lambda_\gamma]_j = [\Delta^k \lambda_\gamma^j]_R$.

Super-CT Construction Phase: For each $j \in \mathcal{S}_g, P_g$ constructs c_γ^j for $\gamma \in [4]$, as in $\mathcal{F}_{\mathsf{GC}}$ (Fig 3.3) and outputs $GC^j = \{\{c_\gamma^j\}_{\gamma \in [4]}\}_{\forall \text{ AND gates}} || \{\mathsf{H}(k_{w,0}^g), \mathsf{H}(k_{w,1}^g)\}_{\forall \text{ output wires w}}$.

Figure 3.5: Protocol $\mathsf{Garble}_4()$

Protocol $\mathsf{Eval}_4()$

Inputs: P_5 holds $GC = GC^1 || GC^2 || GC^3 || GC^4$, blinded bit b_w, the corresponding super-key $\{k_{w,b_w}^g\}_{g \in [4]}$ for every input wire w and mask λ_w for every output wire w.
Output: P_5 outputs $y = C(x)$ where x is the actual input or $\perp$.

Evaluation: Evaluation is done topologically. For a gate with input wires u, v and output wire w, P_5 has $(b_u, \{k_{u,b_u}^g\}_{g \in [4]})$, $(b_v, \{k_{v,b_v}^g\}_{g \in [4]})$.
– For XOR gate, P_5 sets $b_w = b_u \oplus b_v$, $\{k_{w,b_w}^g = k_{u,b_u}^g \oplus k_{u,b_v}^g\}_{g \in [4]}$.
– For AND gate, P_5 sets $\gamma = 2b_u + b_v + 1$ and decrypts every CT c_γ^g in the γth super-CT as follows:

$$(\lambda_\gamma^g || \{[\Delta^{g'} \lambda_\gamma]_g\}_{g' \neq g} || k_w^g) := \mathsf{F}_{k_{u,b_u}^g, k_{v,b_v}^g}(j||g) \oplus c_\gamma^g$$

P_5 then computes $b_w = \oplus_{g \in [4]} \lambda_\gamma^g$ and $k_{w,b_w}^g = k_w^g \oplus (\oplus_{g' \neq g} [\Delta^g \lambda_\gamma]_{g'})$.

For an output wire w, P_5 assigns $\mathbf{Y} := \{k^g_{w,b_w}\}_{g\in[4]}$ and checks if the hash on gth key in $\mathbf{Y}$ indeed maps to $\mathsf{H}(k^g_{w,b_w}), g \in [4]$.
Output: P_5 outputs $y_w := b_w \oplus (\oplus_{g\in[4]} \lambda^g_w)$ for every output wire w.

Figure 3.6: Protocol $\mathsf{Eval}_4()$

3.1.3.2 Efficiency of 4DG

Our 4DG is superior to the state-of-the-art [BLO16] computationally while retaining their communication efficiency. Concretely, for 4DG, [BLO16] needs 4 PRF computations per CT of the super-CT whereas our scheme needs 1 PRF computation per CT. Since, the number of PRFs computed depends on the number of parties, this difference is significant for large n. To elaborate, for n-party garbling, [BLO16] needs n PRF computations per CT of super-CT and hence a total of $\mathcal{O}(n^2)$ PRF per super-CT, while our scheme still needs 1 PRF per CT (so total of n PRFs for super-CT), thus saving $\mathcal{O}(n)$ PRF computations over [BLO16]. The player-emulation technique also impacts the performance of [BLO16] concretely, compared to our 4DG– 12 versus 3 for each CT which has 3 copies and thus, 48 versus 12 per super-CT and 192 versus 48 per AND gate.

3.1.3.3 Correctness and Security of 4DG

Lemma 3.1.4. *The protocols* Garble_4 *and* Eval_4 *are correct.*

Proof. To prove the lemma we argue that the super-key encrypted in the super-CT of a row decrypts to the correct super-key when evaluated on the blinded inputs corresponding to that row. Consider an AND gate with input wires u, v and output wire w with corresponding masks λ_u, λ_v and λ_w respectively. Let the blinded inputs b_u, b_v received for evaluation have values $b_u = b_v = 0$. This means $\gamma = 1$ (row 1). We prove that b_w and $\{k^g_{w,b_w}\}_{g\in[4]}$ are correctly computed given b_u, b_v and super-keys $\{(k^g_{u,b_u}, k^g_{v,b_v})\}_{g\in[4]}$. For simplicity we consider $\lambda_w = 0$. The values $b_u = b_v = 0$ imply $x_u = \lambda_u$ and $x_v = \lambda_v$. Since, $\lambda_w = 0$, $\lambda_\gamma = \lambda_1 = \lambda_u\lambda_v$. This means that $\mathsf{g}(\lambda_u, \lambda_v) = \mathsf{g}(x_u, x_v)$ where g is the AND gate function. Thus, the encrypted super-key must be $\{k^g_{w,\mathsf{g}(x_u,x_v)}\}_{g\in[4]}$ as $\Delta^g\lambda_1 = \Delta^g\mathsf{g}(x_u, x_v)$ (thus $\lambda_1 = \mathsf{g}(x_u, x_v)$) for each garbler P_g. Now, we show that on decryption of the super-CT in row $\gamma = 1$, the evaluator obtains $\{k^g_{w,\mathsf{g}(x_u,x_v)}\}_{g\in[4]}$. The plaintext of super-CT of row 1 on unmasking the one-time pad of PRF appears as follows:

$$\begin{aligned}\{\ &(\lambda_1^1||\{[\Delta^{g'}\lambda_1]_1\}_{g'\neq 1}||k^1_{w,0}\oplus[\Delta^1\lambda_1]_1),\\ &(\lambda_1^2||\{[\Delta^{g'}\lambda_1]_2\}_{g'\neq 2}||k^2_{w,0}\oplus[\Delta^2\lambda_1]_2),\\ &(\lambda_1^3||\{[\Delta^{g'}\lambda_1]_3\}_{g'\neq 3}||k^3_{w,0}\oplus[\Delta^3\lambda_1]_3),\\ &(\lambda_1^4||\{[\Delta^{g'}\lambda_1]_4\}_{g'\neq 4}||k^4_{w,0}\oplus[\Delta^4\lambda_1]_4)\ \}\end{aligned}$$

The evaluator computes $b_w = \oplus_{g\in[4]}\lambda_1^g = \mathsf{g}(x_u, x_v)$ and computes the super-key as $\{(k^g_{w,0} \oplus [\Delta^g\lambda_1]_g) \oplus (\oplus_{g'\neq g}[\Delta^g\lambda_1]_{g'})\}_{g\in[4]} = \{k^g_{w,0} \oplus \Delta^g\lambda_1\}_{g\in[4]}$. Since $\Delta^g\lambda_1 = \Delta\mathsf{g}(x_u, x_v)$, the super-key reduces to $\{k^g_{w,\mathsf{g}(x_u,x_v)}\}_{g\in[4]}$ as desired. The correctness for the remaining rows of super-CT and for any choice of λ_w can be proved in a similar way.

□

3.2 Building Blocks for 4PC

3.2.1 Seed-distribution

The starting point of our 4PC protocols is a semi-honest distributed garbling with $\{P_1, P_2, P_3\}$ as garblers and P_4 as evaluator. The final DGC is denoted as $GC = GC^1||GC^2||GC^3$. Since, we have actively corrupt party in *mixed-adversary* model, we need a mechanism to ensure correctness of the DGC. We adopt the technique of seed-distribution as described in 5PC building blocks and modify it for our 4PC (to ensure correctness of DGC in the face of 1 actively corrupt garbler). We assume that the randomness used to construct GC fragment GC^g by the designated garbler (say P_i) is derived from seed s_g. Now, a corrupt P_i could construct a faulty GC^g. SD enables a pair of parties to construct each fragment of DGC and correctness of that fragment is verified by simply checking the equality of the copies. This strategy suffices when at least one of the two seed-owners is not maliciously corrupt and constructs the DGC fragment honestly.

Our SD works as follows: Three seeds $\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3$ are distributed amongst the garblers P_1, P_2, P_3 such that party P_g holds all but seed s_g. For instance, the fragment GC^1 (analogously GC^2 and GC^3) are constructed by two parties P_2, P_3 who hold seed s_1. We denote by $\mathcal{S}_g$, the indices of the seeds held by party P_g as well as the indices of the parties who hold seed s_g i.e. $\mathcal{S}_1 = \{2, 3\}, \mathcal{S}_2 = \{1, 3\}, \mathcal{S}_3 = \{1, 2\}$. We use this same notation for both 5PC and 4PC protocols and the notation must be interpreted based on whether the context is 5PC or 4PC. The formal protocol appears in Fig 3.7. Additionally like in 4DG seed distribution, this technique also maintains input privacy for colluding parties (1 actively corrupt and 1 passively corrupt) since,

(a) for 2 corrupt garblers, all seeds are known to the adversary but the evaluator is guaranteed to be honest; (b) a colluding garbler and the evaluator lack the knowledge of *one* seed, hence the secrets remain hidden from the adversary.

Protocol π_{seedDist}

Notation $\mathcal{S}_1 = \{2, 3\}$, $\mathcal{S}_2 = \{1, 3\}$, $\mathcal{S}_3 = \{1, 2\}$.

Output Party $P_g, g \in [3]$ outputs seed $\mathsf{s}_i, i \in \mathcal{S}_g$.

Seed-setup P_1 samples a random seed s_2 and runs the routine ExtCom with P_3 as receiver. The broadcast-only transcript of ExtCom (hence, the commitment) is available to the remaining parties too. P_1 runs the routine ExtOpen to obtain opening $o[\mathsf{s}]_2$ which is sent privately to P_3. If the opening is invalid, P_3 aborts. Else, P_3 computes s_2 using $o[\mathsf{s}]_2$.

Similar steps are done by P_2 for seed s_3 and P_3 for seed s_1.

Figure 3.7: Protocol π_{seedDist} for SD in 3DG

For our purposes in the mixed model, SD is done by broadcasting commitment on each seed and sending the opening to only the relevant garbler. This is done to resolve a technicality in the proof and the details are made clear in the relevant section.

3.2.2 Attested Oblivious Transfer

The idea of AOT is similar as described in the 5PC building blocks, except that in the 4PC setting, only one attester is needed (in place of 2 as in 5PC) since there is only 1 active corruption and any malicious behaviour by a possibly corrupt sender P_s, attester P_a or receiver P_r is contained by having an honest party who computes the same message. The formal functionality $\mathcal{F}_{\mathsf{3AOT}}$ (scaled to 3 parties) appears in Fig 3.8.

Functionality $\mathcal{F}_{\mathsf{3AOT}}$

P_s, P_r act as sender and receiver respectively, and P_a acts as attester.

- On input message ($\mathtt{Sen}$, m_0, m_1) from P_s, record (m_0, m_1) and send ($\mathtt{Sen}$, m_0, m_1) to P_a and $\mathtt{Sen}$ to the adversary.
- On input message ($\mathtt{Rec}$, b) from P_r, where $b \in \{0, 1\}$, record b and send ($\mathtt{Rec}$, b) to P_a and $\mathtt{Rec}$ to the adversary.
- On input message ($\mathtt{A}$, m_0^a, m_1^a, b^a) from P_a, if ($\mathtt{Sen}$, sid, $*$, $*$) and ($\mathtt{Rec}$, $*$) have not been recorded,

ignore this message; otherwise, record (m_0^a, m_1^a, b^a) and send `A` to the adversary.

– On input message `Output` from the adversary, if $(m_0, m_1, b) \neq (m_0^a, m_1^a, b^a)$, send $(\texttt{Output}, \perp)$ to P_r; else send $(\texttt{Output}, m_b)$ to P_r.

– On input message `abort` from the adversary, send $(\texttt{Output}, \perp)$ to P_r.

Figure 3.8: Ideal Functionality $\mathcal{F}_{\mathsf{3AOT}}(P_s, P_r, P_a)$ for 3DG

3.2.3 The semi-honest 3DG and Evaluation

The semi-honest 4DG scheme and its evaluation protocol can be trivially scaled to 3DG scheme with $\{P_1, P_2, P_3\}$ as garblers and P_4 as evaluator. We assume that all the randomness required by a party for the GC partition GC^g is generated using the random seed s_g. When coupled with seed distribution and party emulation technique, each GC^g is generated by 2 garblers holding s_g, offering security to at most one actively corrupt garbler. The standard OTs are then replaced with AOT with the use of SD. Similar to 4DG, each AOT is run s.t the attesters hold both seeds that the sender and receiver mutually-exclusively hold. Our formal garbling and evaluation protocols appear in Figs 3.9-3.10. Correctness of 3DG scheme follows from the correctness of 4DG scheme (Lemma 3.1.4).

Next, $\mathcal{F}_{\mathsf{3AOT}}$ can be realized in just one round when enabled with SD [CGMV17] since, every AOT is run between a sender P_s and a receiver P_r s.t there exists an attester P_a who possesses the seeds (inputs) of both P_s, P_r. As a result, P_s, P_a can send the commitments of sender OT message to P_r in one round, while P_a sends the opening for choice bit message in the same round. The receiver then verifies the commitments and computes the OT message. Our AOT is secure against 1 active corruption since, malicious behaviour by a possibly corrupt P_s, P_a or P_r is contained by having an honest party who computes the same message.

For clarity, below we demonstrate, how a particular product share λ_{uv}^2 (of $\lambda_u\lambda_v$) is computed by parties in $\mathcal{S}_2$ ($\{P_1, P_3\}$), utilizing AOT and SD. The share λ_{uv}^2 consists of summands as listed in the first column of the table below. We explain how P_1 computes each summand. Except $\lambda_u^2\lambda_v^2$, the remaining summands correspond to cross-terms that P_1 originally obtained via OT either as sender or receiver. Now, all summands that correspond to P_1 enacting a sender ($\lambda_u^2\lambda_v^g, g \neq 2$) can be sampled from s_2, as the sender's share is a random bit. For the summands where P_1 enacts receiver ($\lambda_u^g\lambda_v^2, g \neq 2$), AOT is needed only for the summand, $\lambda_u^1\lambda_v^2$ that involves s_1 which P_1 does not own, while for other terms, P_1 can locally compute its share with the knowledge of both seeds. As for the AOT, P_1 acts as receiver with seed s_2, P_2 acts as sender with seed s_1, and P_3 act as attester with $\{\mathsf{s}_1, \mathsf{s}_2\}$. Similarly, P_3 can compute the summands of λ_{uv}^2 as indicated in the table.

Summand	P_1 : $(\mathsf{s}_2, \mathsf{s}_3)$	P_3 : $(\mathsf{s}_1, \mathsf{s}_2)$
$\lambda_u^2\lambda_v^2$	local	local
$[\lambda_u^2\lambda_v^1]_S$, $[\lambda_u^2\lambda_v^3]_S$	local	local
$[\lambda_u^1\lambda_v^2]_R$	$\mathcal{F}_{\mathsf{4AOT}}(P_2, P_1, P_3)$	local
$[\lambda_u^3\lambda_v^2]_R$	local	$\mathcal{F}_{\mathsf{4AOT}}(P_2, P_3, P_1)$

Protocol $\mathsf{Garble}_3()$

Common Inputs: Circuit C that computes f.

Primitives and Notation: A double-keyed PRF F [BLO16]. $\mathcal{S}_g$ denotes the indices of parties who hold s_g as well as the indices of seeds held by P_g.

Output: Each party $P_g, g \in [3]$ outputs $GC^j, j \in \mathcal{S}_g$ or $\perp$.

Sampling Phase: Each $P_g, g \in [3]$ samples Δ^j from $\mathsf{s}_j, j \in \mathcal{S}_g$. Also, the following is done for each wire w in C corresponding to seed s_j:

- If w is not an output wire of XOR gate, sample λ_w^j and $k_{w,0}^j$ from s_j. Set $k_{w,1}^j = k_{w,0}^j \oplus \Delta^j$.
- If w is an output wire of XOR gate with input wires u, v, set $\lambda_w^j = \lambda_u^j \oplus \lambda_v^j$, $k_{w,0}^j = k_{u,0}^j \oplus k_{v,0}^j$ and $k_{w,1}^j = k_{w,0}^j \oplus \Delta^j$.

The mask and super-key pair for a wire w is defined as $\lambda_w = \oplus_{g\in[4]}\lambda_w^g$ and $\left(\{k_{w,0}^g\}_{g\in[4]}, \{k_{w,1}^g\}_{g\in[4]}\right)$.
Run in parallel for every AND gate in C with input wires u, v and output wire w:

R1: Product Phase I: Define $\lambda_{uv} = \lambda_u\lambda_v = (\oplus_{g\in[3]}\lambda_u^g)(\oplus_{g\in[3]}\lambda_v^g)$. Likewise define $\lambda_{u\overline{v}}, \lambda_{\overline{u}v}, \lambda_{\overline{u}\,\overline{v}}$ that can be derived from shares of λ_{uv}. Each garbler P_g computes λ_{uv}^j of λ_{uv} for every $j \in \mathcal{S}_g$ as below:

- locally compute $\lambda_u^j\lambda_v^j$. For each $k \neq j$, sample $[\lambda_u^j\lambda_v^k]_S$ from seed s_j.
- for every $k \in \mathcal{S}_g, k \neq j$, locally compute $[\lambda_u^k\lambda_v^j]_R = [\lambda_u^k\lambda_v^j]_S \oplus \lambda_u^k\lambda_v^j$ with the knowledge of s_j, s_k.
- To obtain $[\lambda_u^g\lambda_v^j]_R$ from $\mathcal{F}_{\mathsf{3AOT}}$ acting as receiver with input λ_v^j and P_k with only knowledge of s_g (and not s_j) as the sender with inputs $([\lambda_u^g\lambda_v^j]_S,[\lambda_u^g\lambda_v^j]_S \oplus \lambda_u^j)$ derived from s_g. P_l who has knowledge of $\mathsf{s}_g, \mathsf{s}_j$ acts as attester.
- compute $\lambda_{uv}^j = \lambda_u^j\lambda_v^j \oplus (\oplus_{i\neq j}[\lambda_u^j\lambda_v^i]_S) \oplus (\oplus_{i\neq j}[\lambda_u^i\lambda_v^j]_R)$.

Define $\lambda_1 = \lambda_u\lambda_v \oplus \lambda_w, \lambda_2 = \lambda_u\overline{\lambda_v} \oplus \lambda_w, \lambda_3 = \overline{\lambda_u}\lambda_v \oplus \lambda_w, \lambda_4 = \overline{\lambda_u}\,\overline{\lambda_v} \oplus \lambda_w$. Every P_g computes jth share λ_1^j of λ_1 for all $j \in \mathcal{S}_g$ as $\lambda_{uv}^j \oplus \lambda_w^j$. Similarly, it computes the shares for $\lambda_2, \lambda_3, \lambda_4$.

R2: Product Phase II: P_g computes share $[\Delta^j\lambda_\gamma]_j$ (jth additive share) of $\Delta^j\lambda_\gamma$ for every $\gamma \in [4]$ and $j \in \mathcal{S}_g$ as follows:

- locally compute $\Delta^j\lambda_\gamma^j$. For every $k \neq j$, sample $[\Delta^j\lambda_\gamma^k]_S$ from s_j.

– compute $[\Delta^j \lambda_\gamma]_j = \Delta^j \lambda_\gamma^j \oplus_{k \neq j} [\Delta^j \lambda_\gamma^k]_S$.

P_g computes $[\Delta^k \lambda_\gamma]_j$ of $\Delta^k \lambda_\gamma$ for each $k \neq j$, $\gamma \in [4], j \in \mathcal{S}_g$ as:

- For every $k \in \mathcal{S}_g, k \neq j$, compute $[\Delta^k \lambda_\gamma]_j = [\Delta^k \lambda_\gamma^j]_R$ locally from the knowledge of s_j and s_k.
- To obtain $[\Delta^g \lambda_\gamma^j]_R$ from $\mathcal{F}_{\mathsf{3AOT}}$ acting as receiver with input λ_γ^j and with P_k holding only s_g (and not s_j) as sender whose inputs are $[\Delta^g \lambda_\gamma^j]_S$ and $[\Delta^g \lambda_\gamma^j]_S \oplus \Delta^j$ derived from s_g. P_l who has knowledge of $\mathsf{s}_g, \mathsf{s}_j$ acts as attester. Set $[\Delta^g \lambda_\gamma]_j = [\Delta^g \lambda_\gamma^j]_R$.

Super-CT Construction Phase: For each $j \in \mathcal{S}_g, P_g$ constructs c_γ^j for $\gamma \in [4]$, as in $\mathcal{F}_{\mathsf{GC}}$ (Fig 3.3) and outputs $GC^j = \{\{c_\gamma^j\}_{\gamma \in [4]}\}_{\forall \text{ AND gates}} || \{\mathsf{H}(k_{w,0}^g), \mathsf{H}(k_{w,1}^g)\}_{\forall \text{ output wires w}}$.

Figure 3.9: Protocol $\mathsf{Garble}_3()$

Protocol $\mathsf{Eval}_3()$

Inputs: P_4 holds $GC = GC^1 || GC^2 || GC^3$, blinded bit b_w, the corresponding super-key $\{k_{w,b_w}^g\}_{g \in [3]}$ for every input wire w, mask λ_w for every output wire w.

Output: P_4 outputs $y = C(x)$ where x is the actual input or $\perp$.

Evaluation: Evaluation is done topologically. For a gate with input wires u, v and output wire w, P_4 has $(b_u, \{k_{u,b_u}^g\}_{g \in [3]})$, $(b_v, \{k_{v,b_v}^g\}_{g \in [3]})$.

– For XOR gate, P_4 sets $b_w = b_u \oplus b_v$, $\{k_{w,b_w}^g = k_{u,b_u}^g \oplus k_{u,b_v}^g\}_{g \in [3]}$.

– For AND gate, P_4 sets $\gamma = 2b_u + b_v + 1$ and decrypts every CT c_γ^g in the γth super-CT as follows:

$$(\lambda_\gamma^g || \{[\Delta^{g'} \lambda_\gamma]_g\}_{g' \neq g} || k_w^g) := \mathsf{F}_{k_{u,b_u}^g, k_{v,b_v}^g}(j||g) \oplus c_\gamma^g$$

P_4 then computes $b_w = \oplus_{g \in [4]} \lambda_\gamma^g$ and $k_{w,b_w}^g = k_w^g \oplus (\oplus_{g' \neq g} [\Delta^g \lambda_\gamma]_{g'})$.

For an output wire w, P_4 assigns $\mathbf{Y} := \{k_{w,b_w}^g\}_{g \in [3]}$ and checks if the hash on gth key in $\mathbf{Y}$ indeed maps to $\mathsf{H}(k_{w,b_w}^g), g \in [3]$.

Output: P_4 outputs $y_w := b_w \oplus (\oplus_{g \in [3]} \lambda_w^g)$ for every output wire w.

Figure 3.10: Protocol $\mathsf{Eval}_3()$

Part I

Five-Party Computation with Honest Majority

Chapter 4

5PC with Fairness

Relying on pairwise-secure channels, we outline a symmetric-key based 5PC with fairness, tolerating 2 malicious corruptions with performance almost on par with the state-of-the-art [CGMV17] with selective-abort while maintaining a round complexity of 8. Starting with the overview of [CGMV17], we enumerate the challenges involved in introducing fairness into it and then describe techniques to tackle them and ensure robustness of the output phase.

4.1 Technical Overview

4.1.1 Overview of [CGMV17]

In [CGMV17], the garblers perform a one-time SD, which can be used for multiple executions. The evaluator P_5 splits her input additively among P_2, P_3, P_4 who treat the shares as their own input. Garbling is done using the passively secure scheme of [BLO16] topped with the techniques of SD and AOT (Section 3). For the transfer of super-keys wrt every input wire w of each garbler P_g, the remaining garblers send the mask shares not held by P_g ($\lambda_w^j, j \notin \mathcal{S}_g$) on w to P_g who after verifying the shares for correctness (applying the equality check), computes the blinded bit $b_w = x_w \oplus \lambda_w$ (x_w is the input on w). Now, P_g can send 3 out of 4 keys in the super-key for b_w to P_5. However, to enable P_5 learn the fourth key for b_w that corresponds to the seed held by remaining co-garblers, P_g cannot simply send b_w to the co-garblers, as it would leak P_g's input when two of the garblers are corrupt (and hold all seeds and thus the mask λ_w). Hence, [CGMV17] overcomes this subtle case of masked input key as follows. P_g splits b_w as $b_w = \oplus_{l \in [4] \setminus \{g\}} b_l$ and sends each share to exactly one co-garbler. Each co-garbler now sends key for the share she received to P_5 who XORs the 3 key-shares to get the desired 4^{th} key. The property of free-XOR is crucial in ensuring that XOR of key-shares gives the key on blinded input. A breach in the above solution is that P_g colluding with P_5 can learn both super-keys for w leading to multiple evaluations of f. This is captured by the following attack: P_g sets $b_l = 0$, $b_{l'} = 1$ and sends them to co-garblers P_l, $P_{l'}$ respectively. As a result, P_5 receives 0-key from P_l, 1-key from $P_{l'}$ and XOR of these values leaks the global offset and thus both keys corresponding to the seed P_g does not own. Now P_g who already owns 3 seeds can now use both 0-key and 1-key of the 4$^{\text{th}}$ key to obtain multiple evaluations of f. This is tackled by having P_g and one of her co-garblers separately provide additive shares of 0^κ that are XORed with key-shares before sending to P_5. Finally, P_5 assembles the XOR shares and uses the 4$^{\text{th}}$ key for evaluation. On evaluation, P_5 sends the output super key $\mathbf{Y}$ to all garblers, who then compute the output using output mask shares, that are exchanged and verified at the end of garbling phase.

4.1.2 Our Techniques

The prime challenge to introduce fairness in the protocol of [CGMV17] is for the case of a corrupt evaluator, who either sends $\mathbf{Y}$ selectively to garblers or sends an invalid/no $\mathbf{Y}$ after learning the output herself on successful evaluation of DGC. This can be tackled using the following natural techniques in the output phase: (a) The garblers withhold the shares of mask bits on the output wires until a valid output super-key is received from P_5. (b) To further prevent a corrupt P_5 from selectively sending $\mathbf{Y}$ to garblers, we enforce the garbler who received valid $\mathbf{Y}$ from P_5 to, in turn, send the same $\mathbf{Y}$ to her co-garblers. Nevertheless, both the above solutions can lead to unfair scenarios. In solution (a), a corrupt garbler can send an incorrect share of the mask bit on receiving $\mathbf{Y}$, thus creating chaos for the honest receiver who cannot decide the true value, while the corrupt garbler herself learns the output using the shares received from honest co-garblers. In solution (b), two colluding garblers can convince the honest garblers of any $\mathbf{Y}$ using their knowledge of all seeds, even if the honest P_5 aborts during evaluation. This is easily fixable with broadcast, however, without broadcast, a convincing strategy that $\mathbf{Y}$ indeed originated from P_5 is necessary.

We tackle the concerns in solution (a) using the *commit-then-open* technique. In detail, the garblers are forced to commit to the shares of mask bit on each output wire in advance to bar them from sending inconsistent values later and violating fairness. Three copies of each commitment are sent by the *3-parties* who own the corresponding seed which are then compared for correctness by each receiver prior to evaluation. The collision-resistant property of hash is used as a proofing mechanism to tackle the concerns in solution (b). Concretely, P_5 computes hash on a random value proof in the garbling phase and sends the resulting hash, $\mathsf{H}(\mathsf{proof})$ to all garblers who in turn exchange $\mathsf{H}(\mathsf{proof})$ amongst themselves for consistency. The value proof is sent as a proof to the garblers along with $\mathbf{Y}$ post evaluation. This technique is reminiscent of the one used in [BJPR18]. The above techniques ensure that a colluding garbler and P_5 cannot compute the output y without the aid of at least one honest garbler. An honest garbler reveals shares on the mask bits owned by her only on the receipt of valid $(\mathbf{Y}, \mathsf{proof})$ from some party. This handles the concern in solution (b) by ensuring that $\mathbf{Y}$ was not impostered upon by two colluding garblers as they cannot forge a valid proof.

4.2 The construction

We present the formal protocol in Fig 4.1. The garblers perform a one-time SD as in [CGMV17], which can be used for multiple runs. Circuit garbling is done as in Fig 3.5. The input keys sent by garblers define their committed inputs. The case of evaluator's input and transfer of input keys is dealt as in [CGMV17]. In addition, we enforce each garbler to generate commitments on the shares of output wire masks wrt each seed she owns and allow agreement on these commitments by all parties. Also, P_5 samples a random proof and sends $\mathsf{H}(\mathsf{proof})$ to the garblers who agree on the hash value or abort. Then, P_5 evaluates the GC and sends $(\mathbf{Y}, \mathsf{proof})$ to all. Each garbler checks if $(\mathbf{Y}, \mathsf{proof})$ is valid. If so, it sends $(\mathbf{Y}, \mathsf{proof})$ and the openings corresponding to the commitments on mask bit shares of output wires to all. Finally, when a garbler has enough valid openings for commitments on mask bit shares of output wires, she computes the required output.

Protocol $\mathsf{fair5PC}$

Inputs: Party $P_i \in \mathcal{P}$ has x_i.

Common Inputs: The circuit $C(x_1, x_2, x_3, x_4, \oplus_{j\in\{2,3,4\}} x^{5j})$ that computes $f(x_1, x_2, x_3, x_4, x_5)$ and takes x_1, x_2, x_3, x_4 and shares $\{x^{5j}\}_{j\in\{2,3,4\}}$ as inputs, each input, their shares are from $\{0,1\}$ (instead of $\{0,1\}^\ell$ for simplicity) and output is of the form $\{0,1\}^\ell$.

Notation: $\mathcal{S}_i$ denotes indices of the parties who hold s_i as well as indices of the seeds held by P_i.

Output: $y = C(x_1, x_2, x_3, x_4, x_5)$ or $\perp$.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Section 2.2), an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$ (Section 2.2), Garble_4 (Fig 3.5), Eval_4 (Fig 3.6), Collision Resistant Hash H (Section 2.2).

Seed Distribution Phase (one-time): P_g chooses random seed $\mathsf{s}_g \in_R \{0,1\}^\kappa$, and sends s_g to the other two parties in $\mathcal{S}_g$ who in turn exchange with each other and abort if their versions do not match.

Evaluator's Input sharing Phase: P_5 secret shares its input as $x_5 = x^{52} \oplus x^{53} \oplus x^{54}$. P_5 sends x^{5j} to P_j (wlog).

Proof Establishment Phase: P_5 chooses proof from the domain of hash function H, computes and sends $\mathsf{H}(\mathsf{proof})$ to each garbler $P_g, g \in [4]$. P_g in turn sends the copy of $\mathsf{H}(\mathsf{proof})$ received from P_5 to her co-garblers. P_g aborts if $\mathsf{H}(\mathsf{proof})$ received from a co-garbler does not match with her own copy received from P_5. Else, P_g accepts $\mathsf{H}(\mathsf{proof})$ to be the agreed upon hash.

Setup of public parameter for Equivocal Commitment. For $\mathsf{epp}^g, g \in [4]$ of eNICOM, each $P_j, j \in \mathcal{S}_g$ samples epp^{gj} from fresh randomness (not from any of the seeds he holds) and sends to all. P_g additionally samples $\mathsf{epp}^{gl}, l \in [4] \setminus \mathcal{S}_g$ and sends to all. Each party computes $\mathsf{epp}^g = \oplus_{j \in [4]} \mathsf{epp}^{gj}$. $P_l \in \mathcal{P}$ forwards $\mathsf{epp}^g, g \in [4]$ to all. Each $P_i \in \mathcal{P}$ aborts if any of epp^g received mismatch.

Transfer of Equivocal Commitments.

- Each $P_g, g \in [4]$ runs the **Sampling Phase** of $\mathsf{Garble}(C)$ and computes commitments for every circuit output wire w using randomness from $\mathsf{s}_j, j \in \mathcal{S}_g$ as: $\{(\mathsf{c}_w^j, \mathsf{o}_w^j) \leftarrow \mathsf{eCom}(\mathsf{epp}^j, \lambda_w^j)\}_{j \in \mathcal{S}_g}$. P_g sends $\{(\mathsf{epp}^j, \mathsf{c}_w^j)\}_{j \in \mathcal{S}_g}$ to all.
- $P_i \in \mathcal{P}$ aborts if it receives mismatched copies of $(\mathsf{epp}^j, \mathsf{c}_w^j), j \in [4]$ for some output wire w.

Garbling, Masked input bit and Key Transfer Phase.

- For circuit input wire w held by $P_g, g \in [4]$ corresponding to input bit x_w, each $P_l, l \in [4] \setminus \{g\}$ sends $\lambda_w^j, j \in \mathcal{S}_l$ to P_g. P_g aborts if it receives mismatched copies for some λ_w^j. Else, P_g computes $\lambda_w = \oplus_{j \in [4]} \lambda_w^j$ and $b_w = x_w \oplus \lambda_w$. P_g sends $(b_w, \{k_{w,b_w}^j\}_{j \in \mathcal{S}_g})$ to P_5. To send $k_{w,b_w}^j, j \in [4] \setminus \mathcal{S}_g$ (not held by P_g) to P_5, it does the following (The case for the key of $P_5's$ input share if held by P_g is handled similarly):
  - P_g chooses random bits b_l and random $\beta_l \in \{0,1\}^\kappa$ s.t $b_w = \oplus_{l \in [4] \setminus \{g\}} b_l$ and $0^\kappa = \oplus_{l \in [4] \setminus \{g\}} \beta_l$. P_g sends b_l, β_l to P_l.
  - One garbler other than P_g chooses $\delta_l \in \{0,1\}^\kappa$ s.t $0^\kappa = \oplus_{l \in [4] \setminus \{g\}} \delta_l$ and sends δ_l to P_l.
  - P_l sends $K_l = k_{w,b_w^l}^j \oplus \beta_l \oplus \delta_l$ to P_5 who sets $k_{w,b_w}^j := \oplus_l K_l$.
- For input wire w corresponding to P_5's input shares, let $\{k_{w,0}^g, k_{w,1}^g\}_{g \in [4]}$ be the keys derived from seeds $\{\mathsf{s}_g\}_{g \in [4]}$. Each $P_g, g \in [4]$ computes commitments on these as: for $b \in \{0,1\}, j \in \mathcal{S}_g$, $(c_{w,b}^j, o_{w,b}^j) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k_{w,b}^j)$ using pp^j and randomness derived from s_j and sends $\{\mathsf{pp}^j, c_{w,b}^j\}$ to P_5. P_g also sends o_{w,b_w}^j to P_5 if it holds b_w. P_5 aborts if it receives either different copies of commitments or invalid opening for any wire. Otherwise, P_5 recovers the super-keys for b_w, namely, $\{k_{w,b_w}^g\}_{g \in [4]}$. Let $\mathbf{X}$ to be the set of super-keys obtained.
- $\mathsf{Garble}_4(C)$ is run. Each $P_g, g \in [4]$ sends $\{GC^j\}_{j \in \mathcal{S}_g}$ to P_5. If P_5 finds conflicting copies, it aborts.

Evaluation and Output Phase.

- P_5 runs Eval_4 to evaluate GC using $\mathbf{X}$ and obtains $\mathbf{Y}$ and $(y_w \oplus \lambda_w)$ for all output wires w. P_5 sends $(\mathbf{Y}, \mathsf{proof})$ to all.
- For $g \in [4], j \in \mathcal{S}_g$, if k_{w,b_w}^j of $\mathbf{Y}$ for some output wire w does not match with either $(k_{w,0}^j, k_{w,1}^j)$

or the three keys k^j_{w,b_w} in $\mathbf{Y}$ do not map to the same b_w or if proof does not verify with previously received $\mathsf{H}(\mathsf{proof})$, P_g does nothing. Else, P_g sends $(\mathbf{Y}, \mathsf{proof})$ to all other garblers and $\{\mathsf{o}^j_w\}_{j\in\mathcal{S}_g}$ to all. P_5 checks if valid $\{\mathsf{o}^j_w\}_{j\in\mathcal{S}_g}$ received from each P_g. If so, P_5 computes $y_w = (y_w \oplus \lambda_w) \oplus (\oplus_{l\in[4]}\lambda^l_w)$ for output wire w and thus outputs y.

– If received valid $(\mathbf{Y}, \mathsf{proof})$ and $\{\mathsf{o}^j_w\}_{j\in\mathcal{S}_g}$ from a co-garbler P_g, $P_\alpha, \alpha \in [4]$ computes y by unmasking all λ_w. Also, if sent nothing before, send $(\mathbf{Y}, \mathsf{proof})$ to co-garblers, $\{\mathsf{o}^l_w, \mathsf{o}^j_w\}_{l\in\mathcal{S}_\alpha, j\in\mathcal{S}_g}$ to all. If no y computed yet and received valid $(\mathbf{Y}, \mathsf{proof})$, $\{\mathsf{o}^l_w, \mathsf{o}^j_w\}_{l\in\mathcal{S}_\alpha, j\in\mathcal{S}_g}$ from co-garbler P_α (o^j_w was sent by P_g to P_α before), compute y upon unmasking all λ_w. Likewise, if P_5 has not computed y yet and received valid $\{\mathsf{o}^l_w, \mathsf{o}^j_w\}_{l\in\mathcal{S}_\alpha, j\in\mathcal{S}_g}$ from P_α (o^j_w was sent by P_g to P_α before), P_5 computes y by unmasking all λ_w.

Figure 4.1: Protocol $\mathsf{fair5PC}$

The equivocal commitment eNICOM is used to commit on the output mask shares to handle a technicality that arises in the proof. Namely, when one garbler and P_5 are corrupt, the adversary, on behalf of P_5 can decide to abort as late as when $\mathbf{Y}$ needs to be sent to garblers. Hence, the simulator is also forced to act on the adversary's behalf and invoke the functionality after this step. Nevertheless, the simulator needs to simulate the prior rounds with no clue of the output, which includes transfer of DGC, super-keys, commitments on output mask shares. To tackle this, the simulator uses eNICOM to commit to dummy values at the start and later equivocates to output mask shares (set based on the output obtained after invoking the functionality) if the corrupt P_5 sends $\mathbf{Y}$ to at least one honest garbler. Elaborate details are given in Chapter 4.5.

To keep the eNICOM trapdoor hidden from the adversary and available to the simulator, we need it to be distributed among 3 parties. Although convenient, the public parameter for eNICOM cannot be derived from the seeds, as it would trivially arm a corrupt garbler (with the knowledge of 3 seeds) to equivocate. Further, due to the symmetry of eNICOM, equivocation seems infeasible for the simulator if the trapdoor is distributed into only three parts. Hence, we distribute the trapdoor and thus public parameter into four parts (held by three parties) to keep the binding property intact in the real world while allowing the simulator (acting on behalf of 3 honest parties) to perform equivocation. We demonstrate below for each $g \in [4]$, how $\mathsf{epp}^g (= \oplus_{l\in[4]}\mathsf{epp}^{gl})$ for the output mask bits corresponding to s_g is chosen by the parties. We note that we could opt for a random-oracle based scheme and use its programmability to enable equivocality. But this would make the proof rely on non-standard assumption, and not injective one-way functions. Elaborate details about the instantiation are given in Chapter 2.

	P_1	P_2	P_3	P_4
epp^1	epp^{11}, epp^{12}	–	epp^{13}	epp^{14}
epp^2	–	epp^{21}, epp^{22}	epp^{23}	epp^{24}
epp^3	epp^{31}	epp^{32}	epp^{33}, epp^{34}	–
epp^4	epp^{41}	epp^{42}	–	epp^{43}, epp^{44}

4.2.1 Optimizations

We propose the optimizations below to boost the efficiency of fair5PC: all optimizations of [CGMV17] can be applied to our protocol. More concretely, majority of communication in the garbling phase is due to the number of AOT invocations. This is optimized with the use of batch AOTs. Batch AOTs allow the sender to send both commitments while the attesters send only hash on all the commitments. The NICOM instantiation (Chapter 2) based on the ideal cipher model can be used to obtain faster commitments in practice. Each $GC^g, g \in [4]$, is sent by exactly one owner while the rest send only $\mathsf{H}(GC^g)$. P_5 verifies the hash values before evaluation. For implementation purposes alone, eNICOM, NICOM are instantiated with random-oracle based commitment. Also, communication in eNICOM is saved by generating commitment on the concatenation of mask bit shares of all wires rather than on each bit individually.

4.3 Properties

Lemma 4.3.1. *The protocol* fair5PC *is correct.*

Proof. The input of P_5 is well defined by the shares sent to P_2, P_3, P_4. The 3 keys for each input wire owned by the garblers, along with the 4^{th} key sent as XOR shares, define their committed inputs. Evaluation is done on committed inputs. The correctness of $\mathbf{Y}$ and thus y follows from the correctness of garbling and evaluation (Figs 3.5, 3.6). □

Theorem 4.3.2. *Our* fair5PC *protocol consumes at most 8 rounds.*

Proof. The proof establishment phase and setting up of public parameter for eNICOM consume 2 rounds each and can be overlapped. Further, round 1 of these two phases can be overlapped with distribution of P_5's input and round 1 of masked input bit computation and key transfer phase. These together consume a total of 3 rounds. The key transfer is started prior to Garble. More precisely, garbling can begin alongside round 3 of key transfer phase. The transfer of GC and keys to P_5 take 1 round. Finally, evaluation and output phase need at most 3 rounds, thus settling the protocol in 8 rounds. If $\mathbf{Y}$ is received by all honest garblers in round 1 of output phase itself, then 7 rounds suffice. The seed distribution phase is one-time and hence is not counted for round complexity as in [CGMV17]. □

Theorem 4.3.3. *Assuming one-way permutations, the protocol of* fair5PC *securely realizes* $\mathcal{F}_{\mathsf{fair}}$ *(Fig 2.2) in the standard model against a malicious adversary that corrupts at most two parties.*

The correctness and security proofs appear in Section 4.5.

While the formal security proof is elaborated in Section 4.5, we give the intuition of fairness for completeness. For fairness, we need to guarantee that if the adversary learns the output, then so do honest parties and converse. We first argue in the forward direction. Suppose an adversary gets the output. We consider two corruption cases: Firstly, when P_1 and P_5 are corrupt, the adversary obtains the output only if at least one honest garbler say P_2 receives a valid $(\mathbf{Y}, o)$ from P_5 or P_1 (valid shares of output wire mask bits also from P_1). P_2 sends the received message along with the masking bit shares she owns to all, allowing other parties to compute the output. The recipient garblers further send out their valid masking bit shares to allow any residual party to compute the output. Secondly, when two garblers P_1, P_2 are corrupt, an honest P_5 sends $(\mathbf{Y}, o)$ to all, on successfully evaluating GC. P_1, P_2, knowing all the seeds, can construct the output themselves. The honest garblers send the masking bit shares they hold to all. Thus, every party obtains the output in both cases.

To prove the converse case, suppose the honest parties get the output. We consider the same corruption cases as above. In the first case, it must be true that at least one of the honest garblers say P_2, received a valid $(\mathbf{Y}, o)$ who then sends the masking bit shares it owns along with $(\mathbf{Y}, o)$ to all. Thus, the honest recipients compute the output using $(\mathbf{Y}, o)$ and the masking bit shares from P_2. If P_2 received $\mathbf{Y}$ from P_5, then P_2 uses the masking bit shares sent by P_3, P_4 (once they obtain output) to compute y. Else, P_2 must have received valid $(\mathbf{Y}, o)$ and the masking bit shares from P_1, which is sufficient to compute y. For the case of corrupt P_1, P_2, suppose P_5 gets the output. This implies that all garblers must have obtained the output using valid $(\mathbf{Y}, o)$ sent by P_5 and the masking bit shares received from co-garblers. Consequently, P_5 obtains the output using the masking bit shares sent by honest garblers. This summarizes the intuition.

4.4 n-party Extension of fair5PC

The technique of achieving fairness for 5 parties can be extended to n parties tolerating $t < \sqrt{n}$ corruptions by modifying only the output phase of fair5PC (Fig 4.1). The technical overview is elaborated below.

n-party Extension We first recall the conditions involved in seed distribution for n-parties elaborated in [CGMV17] to better understand the extension tolerating $t \approx \sqrt{n}$ corruptions. The seed distribution needs to satisfy the following properties:

Privacy: No $t-1$ garblers should hold all the seeds. This is to ensure input privacy of honest garblers when $t-1$ garblers and the evaluator collude.

Attested OT For each pair of seeds $\mathsf{s}_i, \mathsf{s}_j$, there must be a garbler who holds both $\mathsf{s}_i, \mathsf{s}_j$. This party will act as an attester in the corresponding AOT.

Correctness Every seed should be held by at least $t+1$ garblers. This is necessary for correctness of the computed DGC.

All the above properties collectively imply that for any corruption scenario, the honest garblers together must hold all the seeds. Specifically, from *correctness*: each seed s_i that is supposed to be held by at least $t+1$ garblers is sure to end up in the hands of an honest garbler in the worst case corruption scenario of t corrupt garblers. To achieve fairness for the case of n parties, all steps of the protocol fair5PC remain the same except the **output phase**. For the extension, we consider that $P_1, ..., P_{n-1}$ are garblers and P_n is the evaluator. On a high level, the output phase involves 3 rounds where in round 1, P_n sends $(\mathbf{Y}, \mathsf{proof})$ to all garblers and the remaining two rounds are used to exchange $(\mathbf{Y}, \mathsf{proof})$ with co-garblers and openings for the commitments on mask-shares belonging to output wires with all and thus fairly compute the output.

Each honest party computes the output only if openings for commitments wrt every seed is received by the end of round 3. A naive way to distribute the openings in the last two rounds is to allow an honest garbler to forward the openings possessed by her (and if received any other) when a valid $(\mathbf{Y}, \mathsf{proof})$ is received. This technique however, leads to fairness violation in the following scenario: suppose the evaluator and $t-1$ garblers are corrupt and P_n does not communicate with any honest garbler in round 1, However in round 2, few of the corrupt garblers send $(\mathbf{Y}, \mathsf{proof})$ to one set of honest parties (chosen selectively s.t the openings of this set of honest parties and those held by the adversary are enough to compute the output). These honest parties forward all the accumulated openings in round 3 and thus the adversary gets the output. Further, in round 3, the adversary can also choose to send the openings to the other complementary set of honest parties on behalf of all the corrupt parties who have not sent anything yet, thus ensuring that other complimentary set gets the output while the first set aborts. To tackle this, we impose a restriction on the garbler P_g who communicates for the first time in round 3 of the output phase as: Forward all the openings accumulated until round 2 only if, the openings received in round 2 together with those held by P_g are sufficient to reconstruct the output. This condition eliminates the dependency of P_g on shares received in round 3 to compute the output and ensures that the adversary, in order to compute the output herself, must aid at least one honest party compute the output. Thus, even if one honest party is able to compute the output at the end of round 2, then that honest party releases all the openings in round 3 sufficient to help all honest parties compute the output. This concludes the intuition. The formal protocol is presented in Fig 4.2.

Protocol n-party Fairness

Round 1: The evaluator sends $(\mathbf{Y}, \mathsf{proof})$ to the garblers.
Round 2: If the received $(\mathbf{Y}, \mathsf{proof})$ from the evaluator is valid, each garbler P_g forwards $(\mathbf{Y}, \mathsf{proof})$ and openings for the commitments on output mask shares wrt the seeds she holds.
Round 3: If received valid $(\mathbf{Y}, \mathsf{proof})$ and valid openings from subset of garblers s.t the openings received and the output mask shares already present with party P_α are sufficient to reconstruct λ_w for every output wire w, then P_α computes output y using the output masks. If sent nothing before, P_α forwards $(\mathbf{Y}, \mathsf{proof})$ and the accumulated openings to all.
Local Computation: If no y computed yet and received valid $(\mathbf{Y}, \mathsf{proof})$ and openings from subset of garblers that are sufficient to reconstruct λ_w for every output wire w, then party P_β computes output y using the output masks.

Figure 4.2: Output Phase for n-party fairness

4.5 Security Proof of fair5PC

We now outline the complete security proof of Theorem 4.3.3 that describes the security of the fair5PC protocol relative to its ideal functionality in the standard security model.

Proof. We describe the simulator $\mathcal{S}_{\mathsf{fair5PC}}$ for the following two cases: First, when two garblers say P_1 and P_2 are corrupt. Second, when one garbler say P_1 and the evaluator P_5 are corrupt. The simulator acts on behalf of all the honest parties in the execution. The corruption of any two garblers is symmetric to the case when P_1, P_2 are corrupt and the corruption of any one garbler and evaluator corrupt is symmetric to the case of P_1, P_5 corrupt.

We briefly highlight the need for equivocal commitment scheme (eNICOM) for the shares of output masking bits in our fair protocol as follows: The adversary can decide to abort the execution as late as when $\mathbf{Y}$ needs to be sent (in the worst case). Consequently, this enforces the simulator to make this decision on behalf of the adversary at the end of Round 5 when calling the functionality. Hence, the simulator needs a mechanism to simulate the earlier rounds appropriately such as sending the GC and committing to the shares of the output masking bits, without the knowledge of whether the execution will result in a valid output or not (with no information about the output). The sending of distributed GC is handled as in any standard distributed garbling proof. To tackle the commitment on shares of output masking bits, the simulator commits to dummy bits for the seed completely under its control. At a later point if the execution results in invoking $\mathcal{F}_{\mathsf{fair}}$ and obtaining y, the simulator equivocates the commitments to desired share bits such that each output wire w decodes to correct y_w. The trapdoor and public parameter for our eNICOM scheme are derived from relevant seeds as described in the protocol.

We provide a high level view of the simulation in distributed garbling and evaluation for completeness. First, in the case of corrupt P_1^*, P_2^*, the evaluator is honest. Hence correctness is required from the DGC. The simulator behaves as an honest $P_i, i \in \{3, 4\}$ following the protocol steps and instructing the functionality to abort in case of any cheating throughout the garbling since all seeds are known to the adversary. If no cheating is detected throughout the DGC construction, then the GC is generated as per the Garble_4 procedure. The inputs of corrupt parties are extracted during the garbled input communication. The simulator sends abort to the functionality if the GC partition sent by P_1^*, P_2^* is not same as the one generated by honest parties.

Second, in the case of corrupt P_1^*, P_5^*, the simulator knows the seeds held by the adversary. In addition the simulator has complete control over the part of GC generated using seed s_2. Since the simulator does not know the output in advance, the masking bit share λ_w^2 corresponding to output wires w cannot be set in advance. As a result, a fake GC is constructed using s_2 that always evaluates to the same output super-key for the extracted and random inputs that are known to the simulator. If the evaluation goes through and $\mathbf{Y}$ is received on behalf of the honest parties, then the simulator invokes the functionality to obtain y, aptly programs the masking bit share under its control by setting $\lambda_w^2 = y \oplus (\oplus_{i \in [4]}, i \neq 2) \lambda_w^i$ for each output wire, performs equivocation on the commitment made for share λ_w^2 and sends the corresponding decommitment to the corrupt parties thus completing simulation. We describe the simulator steps in detail in Figures 4.3, 4.4.

Simulator $\mathcal{S}_{\mathsf{fair5PC}}^{12}$

$\mathcal{S}_{\mathsf{fair5PC}}^{12}$ **(P_1^*, P_2^* are corrupt)**

Seed Distribution Phase (one-time):

– Receive $\mathsf{s}_g, g \in [2]$ from P_g^* on behalf of both P_3, P_4. If the copies of s_g received mismatch, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_g^* and set $y = \perp$.

– Sample random $\mathsf{s}_3, \mathsf{s}_4$ and send s_3 to P_1^*, P_2^* on behalf of P_3 and s_4 on behalf of P_4 to P_1^*, P_2^*.

Evaluator's Input sharing Phase:

– Sample a random $x^{52} \in \{0,1\}^\ell$ as input share of P_5 and send x^{52} to P_2^* on behalf of P_5.

Proof Establishment Phase:

– Sample proof from the domain of hash function H and send $\mathsf{H}(\mathsf{proof})$ on behalf of P_5 to P_1^*, P_2^*.

– Send $\mathsf{H}(\mathsf{proof})$ on behalf of P_3, P_4 to $P_g^*, g \in [2]$. Also receive $\mathsf{H}(\mathsf{proof})$ from P_g^* on behalf of P_3, P_4. If the received hash value from P_g^* does not match with the hash value $\mathsf{H}(\mathsf{proof})$ that was created originally on behalf of P_5, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_g^* and set $y = \perp$.

Setup of public parameter for Equivocal Commitment.

– For eNICOM, receive $\mathsf{epp}^{jg}, g \in [2], j \in \mathcal{S}_g$, $\mathsf{epp}^{gl}, l \in [4] \setminus \mathcal{S}_g$ from P_g^* on behalf of the honest parties. Also send $\mathsf{epp}^{ji}, i \in \{3,4\}, j \in \mathcal{S}_i$, $\mathsf{epp}^{il}, l \in [4] \setminus \mathcal{S}_i$ on behalf of P_i to each P_g^*. Compute $\mathsf{epp}^\alpha = \oplus_{j \in [4]} \mathsf{epp}^{\alpha j}, \alpha \in [4]$ based on the values received from P_g^*. If epp^g does not match with the $\mathsf{epp}^\beta = \oplus_{j \in [4]} \mathsf{epp}^{\beta j}$ computed on behalf of the honest parties, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_g^* and set $y = \perp$. Else forward $\mathsf{epp}^i, i \in [4]$ to P_1^*, P_2^* on behalf of the honest parties.

Transfer of Equivocal Commitments.

– For each circuit output wire w, create equivocal commitments for masking bit shares as per the protocol. Send $\{(\mathsf{epp}^j, \mathsf{c}_w^j)\}_{j \in \mathcal{S}_i}$ on behalf of $P_i, i \in \{3,4\}$ to P_1^*, P_2^*. Also, receive $\{(\mathsf{epp}_w^l, \mathsf{c}_w^l)\}_{l \in \mathcal{S}_g}$ from $P_g^*, g \in [2]$ on behalf of the honest parties. For any output wire w, if the received $(\mathsf{epp}^l, \mathsf{c}_w^l)$ from P_g^*, does not correspond to the one generated using s_l, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_g^* and set $y = \perp$.

Garbling, Masked input bit and Key Transfer Phase.

– For circuit input wires w corresponding to input $x_i i \in [2]$ held by P_i^*, send $\lambda_w^l, l \in \mathcal{S}_j$ on behalf of $P_j, j \in \{3,4\}$ to P_i^*. Similarly, for input corresponding to honest P_j, receive $\lambda_w^l, l \in \mathcal{S}_i$ from P_i^* on behalf of P_j. Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_i^* and set $y = \perp$ if λ_w^l received from P_i^* corresponding to P_j's share does not correspond to the one generated using s_l.

– Sample random bits b_1, b_2 for input wires w of honest $P_i, i \in \{3,4\}$ (including the shares of P_5 that P_i should hold). Send b_1, b_2 to P_1^*, P_2^* respectively on behalf of P_i. For the masked input b_w on wire w of $P_j^*, j \in [2]$, perform the steps as per the protocol to compute $K_l, l \in [4] \setminus \{j\}$.

– For every input wire w belonging to P_5's input share, where $\{k_{w,0}^g, k_{w,1}^g\}_{g \in [4]}$ denote the super-key derived from seeds $\{\mathsf{s}_g\}_{g \in [4]}$, receive $\{c_{w,b}^j\}_{b \in \{0,1\}}$ sent by $P_i^*, i \in [2] \cap \mathcal{S}_l$ on behalf of P_5. If the commitment received for any w from P_i^* does not match with the one originally created, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_i^* and set $y = \perp$.

– For simulation of Round 1 of Garble_4, it is necessary to ensure correctness of the circuit. Behave as honest $P_l, l \in \{3,4\}$ using the seeds chosen in Round 1 and instruct the functionality to abort in case of any cheating detected on behalf of honest P_l based on the messages sent by $P_i^*, i \in [2]$. If an instance of $\mathcal{F}_{\mathsf{4AOT}}$ returns $\perp$ (due to inconsistent messages from $P_i^*, i \in [2]$), then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_i^* and set $y = \perp$.

– For simulation of Round 2 of Garble_4, behave as honest $P_l, l \in \{3, 4\}$. If an instance of $\mathcal{F}_{\mathsf{4AOT}}$ returns $\perp$ (due to inconsistent messages from $P_i^*, i \in [2]$) or $i \in \mathcal{S}_j$ for some $j \in [4]$ sends different GC^j, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_i^* and set $y = \perp$. If there is no abort, then the garble circuit (described in 3.5) will be the output of honest parties.

– Input x_i of $P_i^*, i \in [2]$ is extracted by unmasking λ_w from $b_w = x_i \oplus \lambda_w$ (sent to P_5) for each wire w corresponding to the input of P_i^*. Invoke $\mathcal{F}_{\mathsf{fair}}$ with (Input, x_1), (Input, x_2) to get the output y.

Evaluation and Output Phase.

– Compute $\mathbf{Y}$ such that for all output wires w, each key in $\mathbf{Y}$ maps to $(y_w \oplus \lambda_w)$. Send $(\mathbf{Y}, \mathsf{proof})$ to $P_i^*, i \in [2]$ on behalf of P_5.

– Send $(\mathbf{Y}, \mathsf{proof}, \mathsf{o}_w^j), j \in \mathcal{S}_l$ for all output wires w on behalf of $P_l, l \in \{3, 4\}$ to $P_i^*, i \in [2]$. Also, receive the openings sent by P_g^* similarly. This completes the simulation.

Figure 4.3: Simulator $\mathcal{S}_{\mathsf{fair5PC}}^{12}$ for $\mathsf{fair5PC}$ with actively corrupt P_1^*, P_2^*

The hybrid arguments are as follows:

Security against corrupt P_1^*, P_2^*: We now argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{fair5PC}}^{12}} \overset{c}{\approx} \textsc{real}_{\mathsf{fair5PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1, P_2. The views are shown to be indistinguishable via a series of intermediate hybrids.

– $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{fair5PC}, \mathcal{A}}$.

– $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except that P_5 aborts if any decommitment for $\{k_{w,0}^g, k_{w,1}^g\}_{g \in [4]}$ corresponding to a committed share x^{52} opens to a value other than what was originally committed and held by P_2^*.

– $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except that $\mathbf{Y}$ is computed as $\mathbf{Y} = \{k_{w, y_w \oplus \lambda_w}^g\}_{g \in [4]}$ for each output wire w instead of running the Evaluation Phase of garbling.

– $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that $P_i, i \in \{3, 4\}$ outputs $\perp$ if distributed GC cannot be successfully evaluated by P_5.

$\textsc{hyb}_3 = \textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{fair5PC}}^{12}}$.To sum up the proof, we show that each pair of hybrids is computationally indistinguishable as follows:

$\textsc{hyb}_0 \overset{c}{\approx} \textsc{hyb}_1$: The primary difference between the hybrids is that in $\textsc{hyb}_0$, P_5 aborts if the decommitments sent by P_2 corresponding to the share x^{52} output $\perp$ whereas in $\textsc{hyb}_1$, P_5 aborts if the decommitments sent by P_2^* open to any value other than what was originally committed. Since the commitment scheme Com is strong binding , P_2 could have decommitted successfully to a different valid input label than what was originally committed, only with negligible probability.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that, in HYB_2, $\mathbf{Y}$ is computed as $\mathbf{Y} = \{k^g_{w,y_w \oplus \lambda_w}\}_{g \in [4]}$ instead of running the Evaluation Phase of the garbling. The indistinguishability follows from the correctness of the garbling scheme since $\mathbf{Y}$ computed using $\mathbf{Y} = \{k^g_{w,y_w \oplus \lambda_w}\}_{g \in [4]}$ is equivalent to that computed using the standard Evaluation Phase of garbling.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The only difference between the hybrids is that in HYB_2, $P_i, i \in \{3, 4\}$ can possibly output y which is non-$\perp$ in case it receives a valid proof' such that $\mathsf{H}(\mathsf{proof}') = \mathsf{H}(\mathsf{proof})$ from P_1^* or P_2^* although P_5 was unable to evaluate the GC successfully, whereas in HYB_3, P_i outputs $\perp$ in this case. Due to the collision resistant property of the hash function, P_1^*/P_2^* could have a proof' that can be valid pre-image of $\mathsf{H}(\mathsf{proof})$ only with negligible probability.

Simulator $\mathcal{S}^{15}_{\mathsf{fair5PC}}$

$\underline{\mathcal{S}^{15}_{\mathsf{fair5PC}}\ (P_1^*, P_5^*\ \textbf{are corrupt})}$

Seed Distribution Phase (one-time):

– Receive s_1 from P_1^* on behalf of both P_3, P_4. If the copies of s_1 received mismatch, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_1^* and set $y = \perp$.

– Sample random $\mathsf{s}_3, \mathsf{s}_4$ and send s_3 to P_1^* on behalf of P_3 and s_4 on behalf of P_4.

Evaluator's Input sharing Phase:

– Receive x^{52}, x^{53}, x^{54} on behalf of P_2, P_3, P_4 respectively. Compute $x_5 = \oplus_{j \in \{2,3,4\}} x^{5j}$.

Proof Establishment Phase:

– Receive $\mathsf{H}(\mathsf{proof})$ on behalf of $P_i, i \in \{2, 3, 4\}$ from P_5^*. If the received copies of $\mathsf{H}(\mathsf{proof})$ are not consistent, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_5^* and set $y = \perp$.

– Send $\mathsf{H}(\mathsf{proof})$ to P_1^* on behalf of P_i. Also receive $\mathsf{H}(\mathsf{proof})$ from P_1^* on behalf of P_i. If the copy of the hash value sent by P_1^* is not consistent from that sent by P_5^*, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_1^* and set $y = \perp$.

Setup of public parameter for Equivocal Commitment.

– For eNICOM, receive $\mathsf{epp}^{j1}, j \in \mathcal{S}_1$, epp^{12} from P_1^* on behalf of the honest parties. Also send

$\mathsf{epp}^{ji}, i \in \{2,3,4\}, j \in \mathcal{S}_i$, $\mathsf{epp}^{il}, l \in [4] \setminus \mathcal{S}_i$ on behalf of P_i to each $P_g^*, g \in \{1,5\}$. Compute $\mathsf{epp}^l = \oplus_{j\in[4]}\mathsf{epp}^{lj}, l \in [4]$ based on the values received from P_1^*. If epp^g does not match with the $\mathsf{epp}^\alpha = \oplus_{j\in[4]}\mathsf{epp}^{ij}, \alpha \in [4]$ computed on behalf of the honest parties, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_g^* and set $y = \perp$. Else forward $\mathsf{epp}^i, i \in [4]$ to P_1^*, P_5^* on behalf of the honest parties.

Transfer of Equivocal Commitments.

– For each circuit output wire w, create commitments for masking bit shares known to P_1^* as per the protocol (for $\lambda_w^i, i \in [4] \setminus \{2\}$). Create a dummy commitment c_w^2 for each λ_w^2. Send $\{(\mathsf{epp}^j, \mathsf{c}_w^j)\}_{j\in\mathcal{S}_l}$ on behalf of $P_l, l \in \{2,3,4\}$ to P_1^*, P_5^*. Also, receive $\{(\mathsf{epp}^j, \mathsf{c}_w^j)\}_{j\in\mathcal{S}_1}$ from P_1^* on behalf of the honest parties. If for any j, the received $(\mathsf{epp}^j, \mathsf{c}_w^j)$ from P_1^*, does not correspond to the one generated using s_j, then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_1^* and set $y = \perp$.

Garbling, Masked input bit and Key Transfer Phase.

– For circuit input wires w corresponding to input x_1 held by P_1^*, send $\lambda_w^l, l \in \mathcal{S}_j$ on behalf of $P_j, j \in \{2,3,4\}$ to P_1^*. Similarly, for input corresponding to honest P_j, receive $\lambda_w^l, l \in \mathcal{S}_1$ from P_1^* on behalf of P_j. Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_1^* and set $y = \perp$ if λ_w^l received from P_1^* corresponding to P_j's share does not correspond to the one generated using $\mathcal{S}_1$.

– Sample random b_1 for input wires w of honest $P_i, i \in \{2,3,4\}$ (including the shares of P_5 that P_i should hold). Send b_1 to P_1^* respectively on behalf of P_i. For P_1^*'s input, perform the steps as per the protocol to compute $K_l, l \in \{2,3,4\}$. Send K_l to P_5^* on behalf of P_l. Extract P_1^*'s input x_1 by XORing for each wire w as follows : $x_i = (b_2 \oplus b_3 \oplus b_4) \oplus \lambda_w$ (λ_w is known since all seeds are known).

– For every input wire w belonging to P_5's input share, where $\{k_{w,0}^g, k_{w,1}^g\}_{g\in[4]}$ denote the super-key derived from seeds $\{\mathsf{s}_g\}_{g\in[4]}$, each $P_l, l \in \{3,4\}$ computes commitments on these as per the protocol steps for seeds $\mathsf{s}_3, \mathsf{s}_4$. For commitments in $(c_{w,0}^j, c_{w,1}^j)$ obtained using s_2 that correspond to input labels, generate commitments to the committed shares as per NICOM. Commit to dummy values for all other labels that are not input labels. Send $\{c_{w,b}^i\}_{b\in\{0,1\}, i\in\mathcal{S}_\alpha}$ on behalf of $P_\alpha, \alpha \in \{2,3,4\}$ to P_5^*.

– For simulation of Round 1 of Garble_4 on behalf of honest $P_l, l \in \{2,3,4\}$, all the seeds are known. Additionally, s_2 is not known to P_1^*, so the randomness and garble circuit generated using s_2 is unknown to P_1^*. Participate in the distributed garbling as before but constructing a simulated GC with the help of s_2 such that each ciphertext is encrypts the same output key that represents the masked output which corresponds to the evaluation performed using the extracted inputs of the adversary and the random inputs chosen during simulation. Simulate each instance of $\mathcal{F}_{\mathsf{4AOT}}$ by acting as honest party. If a $\mathcal{F}_{\mathsf{4AOT}}$ instance returns $\perp$ (due to inconsistent messages from P_1^*), then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_1^* and set $y = \perp$.

– For simulation of Round 2 of Garble_4 on behalf of honest $P_l, l \in \{2,3,4\}$, participate in the distributed garbling as described before in round 1 (same strategy as described in [CGMV17]). If an instance of $\mathcal{F}_{\mathsf{4AOT}}$ returns $\perp$ (due to inconsistent messages from P_1^*), then invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of P_1^* and set $y = \perp$. If there is no abort, then the garble circuit (described in Fig 3.5) will be the output of honest parties.

Evaluation and Output Phase.

– Receive $(\mathbf{Y}, \mathsf{proof})$ from P_5^* on behalf of $P_j, j \in \{2,3,4\}$.

– If received $(\mathbf{Y}, \mathsf{proof})$ on behalf of $P_l, l \in \{2,3,4\}$ from P_5^* is such that $\mathbf{Y}$ is same as the output label created in the generation of simulated GC, then invoke $\mathcal{F}_{\mathsf{fair}}$ with (Input, x_1), (Input, x_5) to get the output y and for all output wires w, set $\lambda_w^2 = ((y \oplus \lambda_w) \oplus \lambda_w^j)_{j \in \mathcal{S}_1}$, send $(\mathbf{Y}, \mathsf{proof}, \mathsf{o}_w^j), j \in \mathcal{S}_l$ on behalf of P_l to P_1^* and $(\mathsf{o}_w^j)_{j \in \mathcal{S}_l}$ to P_5^* where $\mathsf{o}_w^2 = \mathsf{Equiv}(\mathsf{c}_w^2, \mathsf{o}_w'^2, \lambda_w^2, t)$ where t is the trapdoor for the commitment c_w^2.

– Else if, received $(\mathbf{Y}, \mathsf{proof}, \mathsf{c}_w^j), j \in \mathcal{S}_1$ on behalf of $P_l, l \in \{2,3,4\}$ from P_1^* (and not from P_5^*), perform checks as per the protocol. If valid, then invoke $\mathcal{F}_{\mathsf{fair}}$ with (Input, x_1), (Input, x_5) and obtain the output y. Send $(\mathsf{c}_w^i, \mathsf{c}_w^j), i \in \mathcal{S}_l, j \in \mathcal{S}_1$ on behalf of P_l to P_5^* where $\mathsf{o}_w^2 = \mathsf{Equiv}(\mathsf{c}_w^2, \mathsf{o}_w'^2, \lambda_w^2, t)$ where t is the trapdoor for the commitment c_w^2.

Figure 4.4: Simulator $\mathcal{S}^{15}_{\mathsf{fair5PC}}$ for $\mathsf{fair5PC}$ with actively corrupt P_1^*, P_5^*

Security against corrupt P_1^*, P_5^*: We now argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^{15}_{\mathsf{fair5PC}}} \overset{c}{\approx} \textsc{real}_{\mathsf{fair5PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1, P_5. The views are shown to be indistinguishable via a series of intermediate hybrids.

– $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{fair5PC}, \mathcal{A}}$.

– $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except that some of the commitments of input wire labels sent by P_2, P_3, P_4 wrt seed s_2, which will not be opened are replaced with commitments of dummy values. These commitments correspond to the labels that do not correspond to any input share.

– $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except that the GC is created as simulated one with the knowledge of s_2.

– $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that,

  - $\textsc{hyb}_{3.1}$: When the execution results in `abort`, the commitment to λ_w^2 for each output wire w is created for a dummy value.
  - $\textsc{hyb}_{3.2}$: When the execution results in output y, the commitment c_w^2 for each output wire w is created for a dummy value and later equivocated to λ_w^2 using o_w^2 computed via where $\mathsf{o}_w^2 = \mathsf{Equiv}(\mathsf{c}_w^2, \mathsf{o}_w'^2, \lambda_w^2, t)$ where t is the trapdoor for the commitment c_w^2.

– HYB_4: Same as HYB_3 except that that the protocol results in abort if the received $\mathbf{Y}$ does not correspond to the $\mathbf{Y}$ resulting from the simulated GC.

$\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^{15}_{\mathsf{fair5PC}}}$. To conclude the proof we show that every consecutive pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference between the hybrids is that some of the commitments of the input labels in HYB_0 corresponding to P_5's input shares that will not be opened are replaced with commitments of dummy values in HYB_1. The indistinguishability follows via reduction to the hiding property of Com.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that in HYB_2, the GC is constructed as a simulated one using the seed s_2 instead of a real GC. More concretely, In HYB_1, Rounds 1, 2 are run as per Garble_4 procedure, which gives $||_{g\in[4]} GC^g$. In HYB_2, it is generated as a simulated circuit such that it always evaluates to the same $\mathbf{Y}$. Indistinguishability follows from the reduction to the security of distributed garbling and in turn the double-keyed PRF F property.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.1}$: The difference between the hybrids is that the commitment to λ^2_w for each output wire w, is created for a dummy value in $\text{HYB}_{3.1}$. The indistinguishability follows via reduction to the hiding property of eCom.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.2}$: The difference between the hybrids is that in $\text{HYB}_{3.2}$, commitment to λ^2_w for each output wire w, is created for a dummy value and later equivocated using o^2_w computed via where $\mathsf{o}^2_w = \mathsf{Equiv}(\mathsf{c}^2_w, \mathsf{o}'^2_w, \lambda^2_w, t)$ where t is the trapdoor for the commitment c^2_w. Indistinguishability follows via reduction to the hiding property of eCom.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The only difference between the hybrids is that, in HYB_3, the protocol aborts if for some output wire w and index $j \in \mathcal{S}_g$, k^j_{w,b_w} of the received $\mathbf{Y}$ does not match with either $(k^j_{w,0}, k^j_{w,1})$ or the keys $\{k^j_{w,b_w}\}_{j\in\mathcal{S}_g}$ in $\mathbf{Y}$ do not map to the same b_w whereas in HYB_4, the protocol results in abort if the received $\mathbf{Y}$ does not match the one created with simulated GC. By security of the garbling scheme, P_5 could have forged such a $\mathbf{Y}$ only with negligible probability. □

Chapter 5

5PC with Unanimous Abort

5.1 Technical Overview and the Construction

By simplifying fair5PC, we present a 5PC achieving unanimous abort, relying on a network of pairwise-private channels with performance on par with [CGMV17] and maintaining the round complexity to 8. Specifically, we eliminate the stronger primitive of eNICOM used to commit on output mask shares in fair5PC, owing to weaker security. However, we still need to address the case of a corrupt P_5 selectively sending $\mathbf{Y}$ to honest garblers. Unanimous abort can be trivially achieved if $\mathbf{Y}$ is broadcast by P_5 instead of being sent privately but since broadcast increases assumptions and is expensive in real-time networks, we enforce the garbler who receives a valid $\mathbf{Y}$ from P_5 to forward the same to her co-garblers. However, this technique does not suffice on its own, since in case of a colluding garbler and the evaluator, P_5 may not send $\mathbf{Y}$ to any honest party and at the same time, the corrupt garbler may send $\mathbf{Y}$ only in the last round, to one honest garbler, thus violating unanimity. To tackle this, we ensure that an honest garbler accepts $\mathbf{Y}$ in the last round of output phase from a co-garbler only if the the co-garbler gives a valid proof that she received $\mathbf{Y}$ from P_5 only in the previous round. This is realized by having each garbler sample a random value and circulate its hash for agreement prior to evaluation of GC. Later in the output phase, if received $\mathbf{Y}$ from P_5, each garbler sends this random value along with $\mathbf{Y}$ to the co-garblers. However, if a garbler P_g who did not receive any message from P_5, receives valid $\mathbf{Y}$ and random value from the co-garbler, then P_g sends her random value along with the $\mathbf{Y}$ and random value of the co-garbler to all. The number of random values received along with $\mathbf{Y}$ from a garbler P_g serve as proof as in which round of output phase P_g received $\mathbf{Y}$. Further, to ensure that $\mathbf{Y}$ indeed originated from P_5 (and was not forged by two corrupt garblers), we reuse the technique described in fair5PC. The formal protocol is presented in Fig 5.1. Similar to our fair protocol, this protocol can also be extended for arbitrary n parties by modifying the output phase of ua5PC (Fig 5.1) as in Fig 5.2.

Protocol ua5PC

Inputs, Common Inputs, Output and Notation : Same as in fair5PC().

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Section 2), Garble_4 (Figs. 3.5), Eval_4 (Fig. 3.6).

Seed Distribution Phase (one-time)and **Evaluator's Input Sharing Phase** are same as in fair5PC().

Proof Establishment Phase: $P_i, i \in [5]$ chooses proof_i from the domain of a hash function H, computes and sends $\mathsf{H}(\mathsf{proof}_i)$ to all parties. Each party, $P_j, j \in [5] \setminus \{i\}$ in turn sends the copy of $\mathsf{H}(\mathsf{proof}_i)$ received to the remaining parties. P_j aborts if the $\mathsf{H}(\mathsf{proof}_i)$ received from the remaining parties does not match with her own copy received from P_i. Else, P_j accepts $\mathsf{H}(\mathsf{proof}_i)$ to be the agreed upon hash.

Setup of public parameter and **Transfer of Equivocal Commitments** are not present in this protocol but instead for each output wire w, each $P_j, j \in \mathcal{S}_g$ sends λ_w^g in clear to all. Each party $P_i \in \mathcal{P}$ aborts if the three copies of λ_w^g received do not match. Else, P_i computes $\lambda_w = \oplus_{g \in [4]} \lambda_w^g$.

Garbling, Masked input bit and Key Transfer Phase are same as in fair5PC().

Evaluation and Output Phase:

- P_5 runs Eval_4 to evaluate GC using $\mathbf{X}$ and obtains $\mathbf{Y}$ and $(y_w \oplus \lambda_w)$ for all output wires w. P_5 sends $(\mathbf{Y}, \mathsf{proof})$ to all. P_5 locally computes $y_w = (y_w \oplus \lambda_w) \oplus_{l \in [4]} \lambda_w^l$ for each output wire w.
- For each $P_g, g \in [4], j \in \mathcal{S}_g$, if the received k_{w,b_w}^j of $\mathbf{Y}$ for some output wire w does not match with either $(k_{w,0}^j, k_{w,1}^j)$ or the three keys $k_{w,b_w}^j, j \in \mathcal{S}_g$ in $\mathbf{Y}$ do not map to the same b_w or proof_5 fails, then do nothing. Else for each output wire w, compute y_w unmasking λ_w. Send $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_g)$ to the co-garblers.
- If received valid $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_g)$ from a co-garbler P_g, $P_\alpha, \alpha \in [4]$ computes y unmasking λ_w. Also if sent nothing before, send $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_g, \mathsf{proof}_\alpha)$ to all. If no output y is computed yet and received valid $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_g, \mathsf{proof}_\alpha)$ from co-garbler P_α (proof_g indicates $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_g)$ was received from P_g), garbler P_γ obtains $(y_w \oplus \lambda_w)$ from $\mathbf{Y}$, unmasks λ_w and computes y.

Figure 5.1: Protocol ua5PC

Optimizations. The efficiency of ua5PC protocol can be boosted similar to fair5PC in both the garbling phase and communication of GC.

5.2 Properties

Lemma 5.2.1. *The* ua5PC *protocol is correct.*

Proof. The input of the evaluator, P_5 is defined to be committed based on the shares sent to P_2, P_3, P_4 in Round 1. The keys communicated by the garblers for their own input define their committed inputs. Evaluation is performed using the committed inputs. The correctness of the output super-key $\mathbf{Y}$ and thus y follows from the correctness of garbling and evaluation (Figs 3.5, 3.6). □

Theorem 5.2.2. *Our* ua5PC *protocol runs in at most 8 rounds.*

Proof. The proof follows from the proof of Theorem 4.3.2. □

Theorem 5.2.3. *Assuming one-way permutations, our protocol* ua5PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{uAbort}}$ *(Fig. 2.3) in the standard model against a malicious adversary that corrupts at most two parties.*

The security proof is provided in Section 5.4.

5.3 n-party Extension of ua5PC

To achieve unanimous abort for the case of n parties, all steps of the protocol ua5PC remain the same except the **output phase**. The seed-distribution is done as explained in Section 4.4. For the extension, we consider that $P_1, ..., P_{n-1}$ are garblers and P_n is the evaluator. On a high level, the output phase involves 3 rounds where in round 1, P_n sends $(\mathbf{Y}, \mathsf{proof}_n)$ to all garblers and the remaining two rounds are used to exchange the $\mathbf{Y}$ and proofs to compute the output.

Each honest party computes the output only if $t+1$ proofs are received by the end of round 3. This is done to prevent the adversary from remaining silent in first two rounds but selectively sending $\mathbf{Y}$ to few honest parties only in the last round and them naively accepting the output without any confirmation about fellow honest parties. Thus, an honest garbler who has not sent anything until the end of round 2, forwards $\mathbf{Y}$ and the received proofs (along with own proof) in round 3 only if at least t valid proofs are received by the end of round 2. This ensures that all honest parties are in agreement about the output acceptance at the end of round 3. In detail, if one honest party decides to accept the output by the end of round 2 due to the availabilty of t proofs, then all honest parties will also accept the output at the end of round 3 due to the availability of at least $t+1$ proofs which implies that an honest party has accepted $\mathbf{Y}$ i round 2. This completes the intuition. We formally present the n-party extension for unanimous abort in Fig 5.2.

Protocol n-party Extension

Let P_n be the evaluator and $P_g, g \in [n-1]$ be the garblers.

Round 1: The evaluator sends $(\mathbf{Y}, \mathsf{proof}_n)$ to the garblers.

Round 2: If the received $(\mathbf{Y}, \mathsf{proof}_n)$ from the evaluator is valid, each garbler P_g forwards $(\mathbf{Y}, \mathsf{proof}_n, \mathsf{proof}_g)$ to all.

Round 3: If received valid $(\mathbf{Y}, \mathsf{proof}_n, \{\mathsf{proof}_g\}_{g \in G})$ where G is a subset of garblers, if the total number of proof_g's and proof_n is at least t, then party P_α outputs y and if sent nothing before, P_α forwards $(\mathbf{Y}, \mathsf{proof}_n, \{\mathsf{proof}_g\}_{g \in G}, \mathsf{proof}_\alpha)$ to all.

Local Computation: If no y output yet and received valid $(\mathbf{Y}, \mathsf{proof}_n, \{\mathsf{proof}_g\}_{g \in G}, \mathsf{proof}_\alpha)$ s.t the total number of proof_g's, proof_n and proof_α together is at least $(t+1)$, then party P_β outputs y using the output super-key and output wire masks for each output wire.

Figure 5.2: Output Phase for n-party unanimous abort

The n-party extension of both ua5PC and fair5PC protocols are designed starting with the n-party extension for selective abort proposed by [CGMV17]. While our ua5PC and fair5PC protocols efficiently achieve UA and fairness respectively against $t \approx \sqrt{n}$ corruptions, there have been prior works in the literature in the honest majority setting $(t < n/2)$ that achieve fairness and GOD [ACJ17],[GLS15],[IKP+16]. However, all these protocols are of theoretical interest and focus on attaining optimal round complexity.

5.4 Security Proof of ua5PC

Proof. We present the proof of Theorem 5.2.3 relative to its ideal functionality $\mathcal{F}_{\mathsf{uAbort}}$ (Figure 2.3). We only outline the sketch of the proof, since it is very similar to the security proof of Theorem 4.3.3, explained in detail in Section 4.5.

We consider two corruption cases: First, when two garblers P_1, P_2 are corrupt and second, when one garbler P_1 and the evaluator P_5 are corrupt. The cases of any two corrupt garblers and one garbler one evaluator corrupt are analogous to the first and second case respectively. The simulator, $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ is described for the first case of corruption as follows: When P_1, P_2 are corrupt, $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ acts on behalf of the honest parties. To begin with, $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ receives $\mathsf{s}_i, i \in [2]$ from P_i^* on behalf of P_3, P_4. If the copies of s_i received mismatch, then $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ invokes the functionality $\mathcal{F}_{\mathsf{uAbort}}$ on behalf of P_i^* with input $\perp$. Else, it samples $\mathsf{s}_j, j \in \{3, 4\}$ and sends s_j to P_1^*, P_2^* on behalf of P_j. A random x^{52} is also sent by $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ on behalf of P_5 to P_2^*. $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ behaves according to the protocol steps in the masked input bit and Key Transfer Phase. The inputs of corrupt parties are extracted similar to our fair protocol. For garbling, since P_1, P_2 are corrupt, correctness must be ensured. $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ behaves as an honest $P_i, i \in \{3, 4\}$ instructing the functionality to abort in case of any cheating during garbling since all seeds are known to the adversary. If no cheating occurs in the GC construction, then a GC is generated as per the Garble procedure. If transfer of keys and masked inputs proceed without any adversarial action, $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ then sends x_1, x_2 to $\mathcal{F}_{\mathsf{uAbort}}$ to obtain y which is the output of GC evaluation. $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ then computes $\mathbf{Y}$ such that for all output wires w, each key in $\mathbf{Y}$ maps to $(y_w \oplus \lambda_w)$. $\mathcal{S}^{12}_{\mathsf{ua5PC}}$ sends continue to $\mathcal{F}_{\mathsf{uAbort}}$ and sends $(\mathbf{Y}, \mathsf{proof}_5)$ on behalf of P_5 and send $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_g)$ on behalf of every honest garbler P_g in the next round to complete the execution.

For the case of a corrupt garbler P_1 and the evaluator P_5, we describe the simulator, $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ as follows: To begin with, $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ receives s_1 from P_1^* on behalf of P_3, P_4. If the copies of s_1 received mismatch, then $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ invokes the functionality $\mathcal{F}_{\mathsf{uAbort}}$ on behalf of P_1^* with input $\perp$. Else, it samples $\mathsf{s}_j, j \in \{3, 4\}$ and sends s_j to P_1^* on behalf of P_j. $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ has the freedom to choose s_2. $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ behaves according to the protocol steps in the masked input bit and Key Transfer Phase. The input of P_5^* is extracted using the shares disclosed by her to the parties with indices in $\{2, 3, 4\}$. The input of P_1^* is extracted in garbled input generation similar to our fair protocol. $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ the invokes the functionality to obtain the output y. Construct a fake garbled circuit using s_2 and the knowledge of y that always evaluates to the same output super-key $\mathbf{Y}$, which corresponds to the evaluation performed using the extracted inputs of the adversary and the inputs of the honest parties. Consequently, the evaluator evaluates the GC to obtain $\mathbf{Y}'$ which is communicated to the garblers. If the labels in $\mathbf{Y}, \mathbf{Y}'$ differ, then $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ instructs the functionality to abort. However, the probability this event is negligible since the adversary can decode only one row of the CT for each gate corresponding to the seed not held by her. This makes the distributions indistinguishable. Finally, if $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ receives a valid pair $(\mathbf{Y}, \mathsf{proof}_5)$ from P_5^* on behalf of honest $P_i, i \in \{2, 3, 4\}$, then $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ sends continue to $\mathcal{F}_{\mathsf{uAbort}}$ and sends $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_i)$ to P_1^* on behalf of P_i. Else if valid $(\mathbf{Y}, \mathsf{proof}_5, \mathsf{proof}_1)$ is received from P_1^* in round 2 of the output phase on behalf of honest P_i, then $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ sends continue to $\mathcal{F}_{\mathsf{uAbort}}$. Else, $\mathcal{S}^{15}_{\mathsf{ua5PC}}$ sends abort to the $\mathcal{F}_{\mathsf{uAbort}}$ to complete the simulation. □

Chapter 6

5PC with GOD

With fair5PC as the starting point, we elevate the security and present a constant-round 5PC with GOD relying only on symmetric-key primitives. We assume a necessary broadcast channel besides pairwise-private channels for our corruption threshold owing to the result of [CL14]. Our protocol reduces to an honest-majority 3PC with GOD in some cases. With the assumption of broadcast channel, our protocol takes 6 rounds when no 3PC is invoked and stretches up to 12 rounds when packed with the 3PC of [BJPR18] in the worst case.

6.1 The Construction

We achieve GOD by tackling the scenarios leading to abort when the parties are in conflict. Specifically, we eliminate a corrupt party and transit to a smaller world of 3 parties with at most one corruption to complete computation in such cases. We retain the setup of four garblers $\{P_1, P_2, P_3, P_4\}$ and P_5 as the evaluator. On a high level, our protocol starts with a robust input and (one-time) SD, followed by the garbling phase, transfer of the GC, blinded inputs and corresponding super-keys to the evaluator and concludes with the circuit evaluation by the evaluator and output computation by all. The key technique in achieving a robust computation lies in the use of tools such as 4-party 2-private RSS and SD to ensure that each phase of the protocol is robust against any malicious wrongdoing. While using a passively-secure 4DG as the underlying building block, there exist scenarios where it seems improbable to publicly identify and eliminate a corrupt party due to the presence of 2 active corruptions. Instead, when the adversary strikes, we establish and eliminate the parties in conflict publicly (of which one is ensured to be corrupt) and rely on the remaining parties with at most one corruption to robustly compute the output. The essence of our protocol lies in tackling the threats to input privacy and correctness that arise during the transfer of masked inputs and corresponding super-keys due to the presence of distinct committees.

To begin with, the input and seed distributions are robust. Each input-share/seed is owned by a committee of 3 parties (as dictated by RSS/seed-distribution). To ensure consistent distribution, we force the dealer (of input-share/seed) to commit to the data publicly and open privately rather than relying on private communication alone. Parties who receive the same RSS share/seed cross-check with each other to agree either on a publicly committed value or a default value when no correct openings are dealt. The shares distributed as per RSS in input distribution are now deemed as parties' new inputs and the circuit is augmented with XOR gates at input level which take these shares as inputs. The formal protocols for input and seed distribution appear in Fig. 6.1 and 6.2 respectively.

Protocol $\mathsf{inputGOD}_i$

Inputs: P_i has input x_i.

Notation: $\mathcal{T}_j, j \in [6]$ denotes the two size maximal unqualified subset ($|\mathcal{T}_j| = 2$) of the parties in the lexicographic order.

Output: Each party $P_k \in \mathcal{P}_i$ outputs $(\mathsf{c}_{ij}, \mathsf{c}'_{ij})_{j\in[6]}$, $\{(\mathsf{o}_{il}, (x^{il} \oplus \mathsf{r}^{il})), (\mathsf{o}'_{il}, \mathsf{r}^{il})\}_{k \notin \mathcal{T}_l \wedge l \in [6]}$ where $(\mathsf{c}_{il}, \mathsf{o}_{il})$, $(\mathsf{c}'_{il}, \mathsf{o}'_{il})$ denote the commitment and opening of the shares $(x^{il} \oplus \mathsf{r}^{il})$, r^{il} respectively.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Chapter 2), a 4-party 2-private RSS.

R1: P_i does the following:

– shares its input as $x_i = \oplus_{j\in[6]} x^{ij}$ and a random input $\mathsf{r}_i \in \{0, 1\}$ as $\mathsf{r}_i = \oplus_{j\in[6]} \mathsf{r}^{ij}$.

– samples pp_i and for $j \in [6]$, computes commitments on $(x^{ij} \oplus \mathsf{r}^{ij})$, r^{ij} as: $(\mathsf{c}_{ij}, \mathsf{o}_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, (x^{ij} \oplus \mathsf{r}^{ij}))$ and $(\mathsf{c}'_{ij}, \mathsf{o}'_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, \mathsf{r}^{ij})$.

– broadcasts $(\mathsf{pp}_i, \mathsf{c}_{ij}, \mathsf{c}'_{ij})$; sends $\{\mathsf{o}_{ij}, \mathsf{o}'_{ij}\}$ privately to each $P_l \notin \mathcal{T}_j$.

Define $\mathcal{X}_{ij}$ to be the set of parties holding the shares $x^{ij} \oplus \mathsf{r}^{ij}$ and r^{ij}. P_i by default belongs to every $\mathcal{X}_{ij}$.

R2: For $\{\mathsf{pp}_i, (\mathsf{c}_{ij}, \mathsf{c}'_{ij})\}_{j\in[6]}$ and $\{\mathsf{o}_{ij}, \mathsf{o}'_{ij}\}$ received from P_i, P_k sets the opening information to $\perp$ when they are invalid and forwards $(\mathsf{o}_{ij}, \mathsf{o}'_{ij})$ to $P_l \notin \mathcal{T}_j$.

Local computation by P_k: P_k resets its opening data on receiving valid openings from fellow parties (if set to $\perp$ earlier). If any opening still remains $\perp$, set agreed-upon default value of $(x^{ij} \oplus \mathsf{r}^{ij})$ and r^{ij}.

Figure 6.1: Protocol $\mathsf{inputGOD}_i$

Protocol $\mathsf{seedGOD}_g$

Notation: $\mathcal{S}_1 = \{1,3,4\}$, $\mathcal{S}_2 = \{2,3,4\}$, $\mathcal{S}_3 = \{1,2,3\}$, $\mathcal{S}_4 = \{1,2,4\}$.

Output: Each party $P_j, j \in \mathcal{S}_g$ outputs s_g.
R1: P_g chooses random seed $\mathsf{s}_g \in_R \{0,1\}^\kappa$, samples pp^g and computes $(\mathsf{c}_g, \mathsf{o}_g) \leftarrow \mathsf{Com}(\mathsf{pp}^g, \mathsf{s}_g)$. P_g broadcasts $(\mathsf{pp}^g, \mathsf{c}_g)$ and sends o_g privately to each $P_j, j \in \mathcal{S}_g$.
R2: If no o_g received or $\mathsf{Open}(\mathsf{pp}^g, \mathsf{c}_g, \mathsf{o}_g) = \perp$, P_j sets $\mathsf{o}_g = \perp$. P_j forwards o_g to $P_k, k \in \mathcal{S}_g$.
(Local Computation by P_j:) Accept o_g sent by P_k, if $\mathsf{Open}(\mathsf{pp}^g, \mathsf{c}_g, \mathsf{o}_g) \neq \perp$ and the o_g received earlier from P_g was set to $\perp$. If the opening still remains $\perp$, agree on default seed s_g.

Figure 6.2: Protocol $\mathsf{seedGOD}_g$

The techniques to identify a pair of conflicting parties (in order to eliminate a corrupt party) differ based on the communication being either *public* or *private*. Public data sent by a party involves the transfer of: (a) GC partition wrt each seed owned by the party, (b) shares of output wire masks wrt each seed owned by the party, (c) shares of input wire masks wrt the seeds not owned by the wire owner, (d) masked input values for the input-shares not owned by the evaluator. Each of these values can be broadcasted by the 3 parties owning the respective seed (for cases (a)-(c)) or input-share (for case (d)). Any mismatch in the 3 broadcasted copies leads to election of a 3-party committee $\mathcal{P}^3$ that becomes the custodian for completing computation. The primary reason for adopting broadcast in the above cases is to aid in unanimous agreement about the conflicting parties. Else, if we rely on private communication alone, an honest receiver may always receive mismatching copies and fail to convince all honest parties about the wrongdoing. Further, input privacy is preserved when masked input is broadcast in case (d) for the shares not owned by evaluator (instead owned by 3 garblers), since the adversary (corrupting the evaluator and one garbler) lacks knowledge of one seed needed to learn the underlying input-share.

Private communication includes the transfer of super-key for input wires wrt masked input shares to P_5. The natural solution is to have the garblers, owning the respective input share, send keys privately to P_5 corresponding to the seeds they own. The private transfer alone, however, allows corrupt parties to send incorrect keys which goes undetected by P_5. We resolve this using the standard trick of *commit-then-open*. All garblers *publicly* commit to both keys on each input wire for the seeds they possess, where any conflict is dealt as in the public message. The commitments wrt each seed are generated by the three seed owners using randomness derived from the same seed, turning public verification to plain equality checking. When no public conflict arises, only the garblers holding the actual input share send the relevant openings to P_5. Since each input-share is owned by at least *two* garblers (the other may be the evaluator), they together hold all parts of the correct super-key to be opened, hence all openings can be communicated. However, this step may not be robust in case of a corrupt garbler sending incorrect (or no) opening privately which can be realised only by P_5. In such case, P_5 raises a conflict against the garbler who sent a faulty opening and a 3-party set is identified for 3PC which excludes P_5 and the conflicting garbler.

Further, input consistency is threatened when the adversary gets the output in the 5PC, yet makes the honest parties receive output via 3PC which now needs to adhere to the inputs committed in the outer 5PC protocol. This occurs when a corrupt P_5 computes the output, yet does not disclose to the garblers and the related 3PC instance invoked must ensure input consistency to bar the adversary from learning multiple evaluations of f. This creates a subtle issue when in the elected 3PC, only one party say P_α holds a share x^{ij} (the other two owners of x^{ij} are eliminated). A potentially corrupt P_α can use a different x^{ij} causing the 3PC to compute on a different input x_i of P_i than what was used in the 5PC, thus obtaining multiple evaluations of f. Custom-made to the robust 3PC of [BJPR18], we tackle this having the RSS dealer P_i distribute $x^{ij} + \mathsf{r}^{ij}$ and r^{ij} instead of just x^{ij} for each share in the input-distribution phase. When a 3PC is invoked, the 3-parties who hold opening of $x^{ij} + \mathsf{r}^{ij}$ and r^{ij} hand them over respectively to the two parties in the 3PC who do not hold x^{ij}. With such a modification, now each input share in the elected 3PC is either held by at least two parties or by one party in which case it is XOR-shared between the remaining two. This is in line with the 3PC of [BJPR18] that offers consistency for inputs, that are either held by at least two parties or by one party in which case it is XOR-shared between the remaining two. In the 3PC of [BJPR18], two parties, say P_α, P_β act as garblers and the third party, say P_γ acts as an evaluator. The garblers use common randomness to construct the same Yao's GC [BHR12] individually. Since at most one party can be corrupt, a comparison of GCs received from the garblers allows P_γ to conclude its correctness. For key transfer, the garblers perform commitments on all keys for the input wires in a permuted order and send openings for the shares they own to P_γ. This suffices since, for an input share not held by P_γ, it is available with both garblers and thus, P_γ can verify if both the openings received for such a share are same. The use of permutation here further ensures that P_γ does not learn the actual value of the input key that she has the opening for. However, for input shares held by P_γ, no permutation is used to allow P_γ to verify if the correct opening has been received. The diagram and an example depicting this process appears in Fig 6.7 (Section 6.4). Our formal 3PC appears in Fig 6.4. The main protocol appears in Fig 6.5.

In 5PC, it is easy to check that the evaluator colluding with a garbler can't cheat with a wrong super-key for the output, as no single garbler possesses all seeds. The AOT protocol, used in Garble, is aptly modified to tackle conflicts and elect a 3PC instance. The protocol realization specific to our 5PC with GOD, god5PC is presented in Fig 6.3. This protocol is same as Π_{4AOT}, except that the sender's and attesters' messages are broadcast to enable the identification of conflict in case of mismatching messages. Thus the protocol either outputs the OT message to the receiver or identifies a 3PC, $\mathcal{P}^3$ for all.

Finally, due to tools customized for 5PC such as RSS, conflict-identification and running smaller 3PC instance, we conclude that our god5PC protocol , in its current form, cannot be extended to n-parties while retaining efficiency, unlike both our fair5PC and ua5PC protocols.

Protocol Π_{4AOTGOD}

P_s, P_r denote the sender and receiver respectively. P_{a_1}, P_{a_2} are attesters. P_a denotes the auditor. All are distinct parties.

Inputs: P_s holds m_0, m_1, P_r holds choice bit b.

Notations $\mathcal{P}^3$ is the 3PC committee with at most 1 corruption.

Output P_r outputs $m_b/\mathcal{P}^3$. All other parties output $\perp/\mathcal{P}^3$.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Section 2.2).

- P_s samples pp and random $r_0, r_1 \leftarrow \{0,1\}^\kappa$ (derived from s_i, $i \in \mathcal{S}_s \setminus \mathcal{S}_r$) and computes $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$, $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_s broadcasts $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$. P_{a_1}, P_{a_2} who know (r_0, r_1) (since they know s_i) also compute $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$, $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$ and each broadcast $(\mathsf{c}_0, \mathsf{c}_1)$.
- P_r has b (derived using $\mathsf{s}_j, j \in \mathcal{S}_r \setminus \mathcal{S}_s$) which is known to P_{a_1}, P_{a_2} (since they know s_j). P_{a_1} (wlog) sends o_b to P_r.

If the broadcast values sent by P_s, P_{a_1}, P_{a_2} do not match, each $P_\gamma, \gamma \in [5]$ sets $\mathcal{P}^3 := \{a_1, r, a\}$. Output $\mathcal{P}^3$.

(Computation by P_r): If no o_b is received or $\mathsf{Open}(\mathsf{c}_b, \mathsf{o}_b) = \perp$, broadcast conflict with P_{a_1}. All parties set $\mathcal{P}^3 := \{s, a_2, a\}$ and output $\mathcal{P}^3$. Else, P_r outputs $m_b = \mathsf{Open}(\mathsf{c}_b, \mathsf{o}_b)$ and the remaining parties output $\perp$.

Figure 6.3: Protocol $\Pi_{\mathsf{4AOTGOD}}(P_s, P_r, \{P_{a_1}, P_{a_2}\}, P_a)$ for god5PC

Protocol god3PC

Inputs: Party P_k has $(\mathsf{c}_{ij}, \mathsf{c}'_{ij})$ for $i \in [5], j \in [6]$ and $(\mathsf{o}_{il}, \mathsf{o}'_{il})$ for $i \in [5], l \in [6], P_k \notin \mathcal{T}_l$.

Common Inputs: The circuit $C(\oplus_{j\in[6]}x^{1j}, \oplus_{j\in[6]}x^{2j}, \oplus_{j\in[6]}x^{3j}, \oplus_{j\in[6]}x^{4j}, \oplus_{j\in[6]}x^{5j})$ that computes $f(x_1, x_2, x_3, x_4, x_5)$, each input, their shares and output are from $\{0, 1\}$.

Notation: $\mathcal{P}^3 = \{P_\alpha, P_\beta, P_\gamma\}$ is the chosen 3PC Committee.

Output: $y = C(x_1, x_2, x_3, x_4, x_5)$.

Input Setup for 3PC: For each x^{ij}, if just one party, say $P_\alpha \in \mathcal{P}^3 \cap \mathcal{X}_{ij}$, the following is done: every party in $\mathcal{X}_{ij}$ sends o_{ij} for $x^{ij} \oplus \mathsf{r}_{ij}$ and o'_{ij} for r_{ij} to P_β and P_γ respectively, each of which in turn recovers the respective share using one valid opening.

3PC Run: Run a robust 3PC (Fig 6.6 [BJPR18] secure against one active corruption with $\{P_\alpha, P_\beta\}$ as garblers and P_γ as the evaluator.

- The input of each party is x^{ij}/ $x^{ij} \oplus \mathsf{r}^{ij}$ / r^{ij}. P_γ does *not* XOR-share its input as in the protocol of [BJPR18].
- Inside the 3PC, for inputs not known to P_γ, the garblers send commitments on both keys in random permuted order with randomness drawn from the common randomness of garblers. For other inputs, the commitments are sent without permutation.
- For x^{ij}, not known to P_γ and held by both P_α, P_β and on receiving the opening for keys P_γ, checks if the opened keys are same from both garblers. For x^{ij} known to P_γ, it checks if they correspond to bit x^{ij} by checking whether x^{ij}th commitment was opened or not.
- The case when all 3 parties hold x^{ij} is subsumed in the above case.
- For x^{ij} held by P_γ while $x^{ij} \oplus \mathsf{r}^{ij}$ and r^{ij} held by P_α and P_β respectively, P_γ (who knows $x^{ij} \oplus \mathsf{r}^{ij}$ and r^{ij} too) checks if the openings obtained from P_α and P_β indeed correspond to $x^{ij} \oplus \mathsf{r}^{ij}$ and r^{ij} respectively. If so, he XORs the keys to obtain the key for x^{ij}.
- For x^{ij} held by P_α, while $x^{ij} \oplus \mathsf{r}^{ij}$ held by P_β and r^{ij} held by P_γ, P_α sends key-openings wrt $x^{ij} + r^{ij}, \mathsf{r}^{ij}$ and P_β sends key-opening wrt $x^{ij} \oplus \mathsf{r}^{ij}$. P_γ checks if the opening wrt r^{ij} is correct and if the opened keys wrt $x^{ij} \oplus \mathsf{r}^{ij}$ (sent by P_α, P_β) are the same. If so, the keys of r^{ij} XORed with $x^{ij} \oplus \mathsf{r}^{ij}$ top obtain key wrt x^{ij}. Compute similarly if $x^{ij} \oplus \mathsf{r}^{ij}$ is held by P_γ.
- The rest of 3PC is run using keys for all RSS shares x^{ij} and the output obtained is sent to each $P_i \in \mathcal{P}$.

Output: The parties output majority of the three y's received.

Figure 6.4: Protocol god3PC

Protocol god5PC

Inputs and Output: Party $P_i \in \mathcal{P}$ has x_i. Each party outputs $y = C(x_1, x_2, x_3, x_4, x_5)$.

Common Inputs: The circuit $C(\oplus_{j\in[6]}x^{1j}, \oplus_{j\in[6]}x^{2j}, \oplus_{j\in[6]}x^{3j}, \oplus_{j\in[6]}x^{4j}, \oplus_{j\in[6]}x^{5j})$ that takes the RSS shares as inputs and computes $f(x_1, x_2, x_3, x_4, x_5)$, each input, their shares are from $\{0,1\}$ (instead of $\{0,1\}^\ell$ for simplicity) and output is from $\{0,1\}^\ell$.

Notation: $\mathcal{S}_i$ denotes the indices of the parties who hold s_i as well as the indices of the seeds held by P_i. $\mathcal{X}_{ij}$ denotes the set of parties that holds the j^{th} share of P_i's input x^{ij}. $\mathcal{P}^3$ is the identified 3PC committee.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Section 2), $\mathsf{inputGOD}_i$ (Fig 6.1), $\mathsf{seedGOD}_g$ (Fig 6.2), Garble_4 (Fig 3.5), Eval_4 (Fig 3.6) and Π_{4AOTGOD} (Fig 6.3).

Input and Seed Distribution Phase. Run $\mathsf{inputGOD}_i$ and $\mathsf{seedGOD}_g$ for every $P_i \in \mathcal{P}$ and $P_g, g \in [4]$ respectively in parallel.

Garbling Phase. $\mathsf{Garble}_4(C)$ is run where Π_{AOTGOD} (Fig 6.3) is used instead of $\mathcal{F}_{\mathsf{4AOT}}$ to achieve OT. Each $P_g, g \in [4]$ broadcasts $\{GC^j\}_{j\in\mathcal{S}_g}$. Each party runs $\mathsf{god3PC}$ with $\mathcal{P}^3$ when any instance of Π_{4AOTGOD} returns $\mathcal{P}^3$ or with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$ when (P_α, P_β) with $\alpha, \beta \in \mathcal{S}_g$ for some $g \in [4]$ broadcasts different GC^g (in the optimized version, we broadcast only a hash of GC).

Masked input bit and Key Transfer Phase.

- In parallel to the **R1** of *Garbling phase*,
  - For each *output* wire w, $P_g, g \in [4]$ broadcasts $\lambda_w^j, j \in \mathcal{S}_g$. Every party runs $\mathsf{god3PC}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$, if parties P_α, P_β holding seed s_g i.e. $\{\alpha, \beta\} \in \mathcal{S}_g$ broadcast different copies of λ_w^g for some output wire w and g. (Tie break deterministically if multiple pairs are in conflict.) Otherwise, every party reconstructs $\lambda_w = \oplus_{g\in[4]}\lambda_w^g$ for every output wire w.
  - For every *input* wire w corresponding to input $x_w = x^{ij}$ held by three garblers, for each $P_g \in \mathcal{X}_{ij}$: each garbler $P_h, h \neq g$, broadcasts $\lambda_w^l, l \in \mathcal{S}_h \setminus \mathcal{S}_g$. (If $\mathcal{X}_{ij}$ includes evaluator, then each garbler $P_h, h \in [4]$ broadcasts $\lambda_w^l, l \in \mathcal{S}_h$). Every party runs $\mathsf{god3PC}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$, if there are parties P_α, P_β with $\{\alpha, \beta\} \in \mathcal{S}_l$ broadcasting different copies λ_w^l for some wire w. Otherwise, P_g, the owner of the input wire w uses λ_w^l to compute $\lambda_w = \oplus_{l\in[4]}\lambda_w^l$.
- In parallel to **R2** of *Garbling phase*, for circuit input wire w corresponding to input $x_w = x^{ij}$ held by three garblers, each $P_\alpha \in \mathcal{X}_{ij}$ computes $b_w = x_w \oplus \lambda_w$ and broadcasts b_w. Every party runs $\mathsf{god3PC}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$, if there are parties P_α, P_β with $\{\alpha, \beta\} \in \mathcal{X}_{ij}$ broadcasting different copies of b_w. Otherwise, P_5 uses $b_w (= x_w \oplus \lambda_w)$ for evaluation. For circuit input wire w corresponding to input $x_w = x^{ij}$ held by two garblers and P_5, P_5 already knows b_w as λ_w was computed by P_5 in the previous step.
- For every input wire w, let $\{k_{w,0}^g, k_{w,1}^g\}_{g\in[4]}$ denote the super-key derived from seeds $\{\mathsf{s}_g\}_{g\in[4]}$. Each $P_g, g \in [4]$ computes commitments as: for $b \in \{0,1\}, j \in \mathcal{S}_g$, $(c_{w,b}^j, o_{w,b}^j) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k_{w,b}^j)$ and broadcasts $\{\mathsf{pp}^j, c_{w,b}^j\}$. P_g sends the opening o_{w,b_w}^j to P_5 if it also holds b_w. Every party runs $\mathsf{god3PC}$ with $\mathcal{P}^3$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$ if (P_α, P_β) with $\alpha, \beta \in \mathcal{S}_i$ for some i and input wire w broadcast different commitments. Otherwise, P_5 tries to recover the super-key for b_w, namely,

$\{k^g_{w,b_w}\}_{g\in[4]}$ using the openings received. If no valid openings received for some key, P_5 broadcasts a conflict with a garbler who sent invalid opening and subsequently every party runs $\mathsf{god3PC}$ with the remaining three parties as $\mathcal{P}^3$. Otherwise, let $\mathbf{X}$ to be the set of super-keys obtained.

Evaluation and Output Phase.

– P_5 runs Eval_4 to evaluate $\mathbf{C}$ using $\mathbf{X}$ and obtains $\mathbf{Y}$ and $(y_w \oplus \lambda_w)$ for all output wires w. For each output wire w, P_5 computes $y_w = (y_w \oplus \lambda_w) \oplus_{g\in[4]} \lambda^g_w$ and thus y. Finally, P_5 outputs y. P_5 broadcasts $\mathbf{Y}$.

– Every party P_g runs $\mathsf{god3PC}$ with $\mathcal{P}^3$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_1, P_5\}$ if k^j_{w,b_w} of $\mathbf{Y}$ for some output wire w and index $j \in \mathcal{S}_g$ does not match with either $(k^j_{w,0}, k^j_{w,1})$ or the three keys $k^j_{w,b_w}, j \in \mathcal{S}_g$ in $\mathbf{Y}$ do not map to the same b_w. Otherwise, each garbler P_g obtains $(y_w \oplus \lambda_w)$ by comparing each key in $\mathbf{Y}$ with the two key labels for each w and computes $y_w = (y_w \oplus \lambda_w) \oplus_{g\in[4]} \lambda^g_w$. Finally, P_g outputs y.

Figure 6.5: Protocol $\mathsf{god5PC}$

6.2 Optimizations

To improve efficiency, the garbling process is optimized similar to $\mathsf{fair5PC}$. When a conflict is identified prior to the sending of GC, identification of the 3PC instance and its execution are set in motion immediately, thus enabling the protocol to terminate faster. To minimize the overhead of broadcast and make it independent of input, output and circuit size, we replace each broadcast message m with the collision-resistant hash of the message, $\mathsf{H}(m)$, while sending m privately to the recipient. For instance, in DGC, $\mathsf{H}(GC^i), i \in [4]$ is broadcasted by parties who own GC^i whereas, GC^i is sent to the evaluator by one of the parties in $\mathcal{S}_i$ privately. Similarly, for sending output super-key, $\mathsf{H}(\mathbf{Y})$ is broadcasted by P_5 and $\mathbf{Y}$ is sent via pairwise channels and so on. With this optimization in broadcast, we elaborate how any conflict will be resolved with the following examples (all our broadcast messages fall under one of these examples):

Example 1: Consider a message m to be broadcasted where m is the GC fragment GC^1. This fragment is held by P_1, P_3, P_4 due to seed distribution. Each of P_1, P_3, P_4 broadcasts $\mathsf{H}(GC^1)$. If the hashes mismatch for two parties say P_1, P_3, then a 3PC instance is formed with P_2, P_4, P_5. Else, if all the broadcast hashes are in agreement, then P_1 will send GC^1 privately to P_5. Now if P_5 is honest and finds that the received GC^1 is not consistent with the hash that was successfully broadcasted and agreed, then P_5 broadcasts a conflict with P_1 and a 3PC instance with P_2, P_3, P_4 is chosen. Else if P_5 is corrupt and raises a false conflict with P_1, even then the 3PC with P_2, P_3, P_4 is run. In both the cases, one corrupt party is surely eliminated and the 3PC contains at most one corruption.

Example 2: Consider a message m to be broadcasted where m is the mask share λ_w^1 on output wire w. The mask-share λ_w^1 is held by P_1, P_3, P_4 due to seed distribution. Each of P_1, P_3, P_4 broadcasts $\mathsf{H}(\lambda_w^1)$. If the hashes mismatch for two parties say P_1, P_3, then a 3PC instance is formed amongst the remaining parties, P_2, P_4, P_5. Else, if all the hashes are in agreement, then P_1, P_3, P_4 privately send λ_w^1 to each party. We consider the receiver P_2 for explanation. This step is robust since if the hashes are in agreement, there will always exist one valid pre-image among the private messages received by P_2. This is because, even if two of the three senders P_1, P_3 are corrupt and send inconsistent preimage, P_4 will send valid λ_w^1 which will be consistent with the agreed upon hash. Hence P_2 uses the value sent by P_4 and proceeds for computation.

6.3 Properties

Lemma 6.3.1. *An elected 3PC has at most one corruption.*

Proof. We argue that a corrupt party is eliminated in a conflict. Suppose P_i, P_j are in conflict. This could be due to either (i) mismatch in the public message broadcast by P_i, P_j or (ii) one of P_i, P_j raised a conflict against the other for an incorrect private message. In case (i), each message is result of either robust input or seed distribution and hence if both were honest, the broadcast messages would be identical. In case (ii), each message involves an opening for the commitments agreed on in public message and neither P_i nor P_j would raise a conflict if valid opening was received. Also, in both the above cases, each message is checked for correctness before proceeding further and thus the conflict could not have been the result of adversary's doing in the previous steps. This implies that at least one of P_i, P_j is corrupt. Thus, an elected 3PC in either case would contain parties $\mathcal{P}^3 = \mathcal{P} \setminus \{P_i, P_j\}$. Since one of P_i, P_j is surely corrupt, at most one corrupt party can be present in $\mathcal{P}^3$. □

Lemma 6.3.2. *The output y computed in the* god3PC *instance corresponds to the committed inputs.*

Proof. In case of conflict in god5PC, a 3PC instance with at most one corruption is formed (Lemma 6.3.1). To ensure input consistency in the 3PC, every agreed upon RSS share x^{ij} in inputGOD, is made available in 3PC to at least two parties or when held by one party, it is XOR shared between the remaining two. With this arrangement of input shares, the robust 3PC of [BJPR18] is guaranteed to preserve input consistency. This ensures that computation in 3PC is performed on the inputs committed in inputGOD. □

Theorem 6.3.3. *The protocol* god5PC *is correct.*

Proof. We argue that the output y computed corresponds to the unique inputs committed by each $P_i, i \in [5]$ in $\mathsf{inputGOD}_i$. A corrupt party either commits to an input or a default value is assumed as per $\mathsf{inputGOD}$. The honest parties are established to have committed to their inputs by the end of round 1 in $\mathsf{inputGOD}$. An honest P_α obtains the output either by decoding the output super-key $\mathbf{Y}$ or via the output of $\mathsf{god3PC}$ (as a participant in $\mathsf{god3PC}$ or recipient from the 3PC committee). In the latter case, correctness follows from Lemma 6.3.2 and correctness of $\mathsf{god3PC}$. We argue for the former case. Let an honest P_α obtains output from $\mathbf{Y}$ broadcast by P_5. This implies that the adversary behaved honestly in the entire execution and the input keys opened by a corrupt garbler correspond to committed inputs only. Otherwise, a conflict would be raised to elect a 3PC, which contradicts our assumption that the output was obtained on decoding $\mathbf{Y}$. Thus, the output always corresponds to the committed inputs in $\mathsf{inputGOD}$. The correctness of evaluation follows from the correctness of the garbling scheme (Figs 3.5, 3.6). □

Lemma 6.3.4. *Assuming a broadcast channel, our protocol* $\mathsf{god5PC}$ *runs in at most 12 rounds.*

Proof. The robust routine $\mathsf{inputGOD}_i$ needs 2 rounds. In the honest run, Garble_4 requires 2 rounds which can be overlapped with transfer of mask bit shares on input wires and output wires publicly. Transfer of input super-keys, blinded inputs and the distributed GC takes 1 round. Finally, 1 last round is required for sending $\mathbf{Y}$ by the evaluator. Thus, 6 rounds suffice for GOD in an honest run.

The worst case run occurs when a corrupt P_5 chooses not to send the output super-key to garblers. In such a case, the round complexity inflates to at most 12, since at most 5 rounds are necessary for the robust 3PC [BJPR18] and 1 extra round to send the output of 3PC instance to all parties. In all other cases of conflict, at most one round is used to establish the conflict and elect the 3PC. Thus, the round complexity in such cases is less than the worst case run. □

Theorem 6.3.5. *Assuming one-way permutations, protocol* $\mathsf{god5PC}$ *securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Fig. 2.1) in the standard model against an active adversary that corrupts at most two parties.*

The security proof is presented in Section 6.6. Since the inputs are defined prior to the garbling phase in $\mathsf{god5PC}$, we do not require the adaptive notion of the proof. The same is true for all our protocols.

Although, the formal security proof appears in Section 6.6, here, we provide intuition of GOD for completeness. The routine $\mathsf{inputGOD}$ binds the adversary to commit to an input or a default value. If a conflict is identified at any point during the execution, then an elected 3PC committee runs robust 3PC of [BJPR18] to obtain the output y. Otherwise, computation proceeds as per the honest run and each party receives the output using the $\mathbf{Y}$ broadcasted by P_5. If $\mathbf{Y}$ is valid, then all parties compute y using $\mathbf{Y}$ to conclude the execution. Else if $\mathbf{Y}$ is invalid or not received, a 3PC instance is identified among the garblers to compute y. In both the above cases (lemma 6.3.3), inputs committed in inputGOD alone are used to obtain the output y thus concluding the intuition.

6.4 3PC with GOD

In this section, we include the robust 3PC instantiation of [BJPR18] verbatim in Fig 6.6 for completeness. For every case of conflict when a 3PC committee is chosen, the routine god3PC invokes the protocol in Fig 6.6 to compute the output robustly while ensuring consistency of inputs committed in inputGOD routine. In the protocol g3PC given below, that is assumed to run between the 3 parties P_1, P_2, P_3, P_1, P_2 act as garblers and P_3 is the evaluator. Yao's garbled circuit [Yao82] with security defined as per [BHR12, LP04] is used for garbling. The property of soft decoding used in this protocol allows decoding of the garbled circuit output without the use of decoding information [MRZ15]. This can be trivially achieved by appending the truth value to each output key.

Protocol g3PC

Inputs: Party P_α has x_α for $\alpha \in [3]$.
Common Inputs: The function $C(x_1, x_2, x_3, x_4)$ that computes $f(x_1, x_2, x_3 \oplus x_4)$ where inputs, function output are in $\{0,1\}^\ell$ for $\ell \in \mathsf{poly}(\kappa)$. P_3 is the evaluator and (P_1, P_2) are the garblers.
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 $s \in_R \{0,1\}^\kappa$ for G and sends s to P_2. Besides,

- 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 $(c_{31}, o_{31}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{31})$, $(c_{32}, o_{32}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{32})$, broadcasts $\{\mathsf{pp}, c_{31}, c_{32}\}$ and sends (x_{31}, o_{31}), (x_{32}, o_{32}) to P_1, P_2 respectively. (This step is not done in our 3PC. as god3PC already does this step to ensure input consistency and privacy).

Round 2: $P_i (i \in [2])$ broadcasts $(\mathtt{conflict}, P_3)$ if $\mathsf{Open}(c_{3i}, o_{3i}) \neq x_{3i}$. Else, it does the following:

- Compute GC $(\mathbf{C}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$ with randomness from $G(s)$. Assume $\{\mathsf{K}^0_\alpha, \mathsf{K}^1_\alpha\}_{\alpha \in [\ell]}$, $\{\mathsf{K}^0_{\ell+\alpha}, \mathsf{K}^1_{\ell+\alpha}\}_{\alpha \in [\ell]}$, $\{\mathsf{K}^0_{2\ell+\alpha}, \mathsf{K}^1_{2\ell+\alpha}\}_{\alpha \in [2\ell]}$ refer to encoding information for the input of P_1, P_2 and shares of P_3 respectively (w.l.o.g).

- Compute permutation strings $p_1, p_2 \in_R \{0,1\}^\ell$ for garblers' input wires, generate commitments on e using randomness from $G(s)$. For $b \in \{0,1\}$, $(c^b_\alpha, o^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p_1^\alpha \oplus b}_\alpha)$, $(c^b_{\ell+\alpha}, o^b_{\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p_2^\alpha \oplus b}_{\ell+\alpha})$ for $\alpha \in [\ell]$, $(c^b_{2\ell+\alpha}, o^b_{2\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, e^b_{2\ell+\alpha})$ for $\alpha \in [2\ell]$. Broadcast $\mathcal{B}_i = \{\mathbf{C}, \{c^b_\alpha\}_{\alpha \in [4\ell], b \in \{0,1\}}\}$.
- 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 $\{o^{m_1^\alpha}_\alpha, o^{x_{31}^\alpha}_{2\ell+\alpha}\}_{\alpha \in [\ell]}$, m_1. Similarly, P_2 computes $m_2 = x_2 \oplus p_2$ and sends to P_3: openings of the commitments corresponding to (x_2, x_{32}) i.e $\{o^{m_2^\alpha}_{\ell+\alpha}, o^{x_{32}^\alpha}_{3\ell+\alpha}\}_{\alpha \in [\ell]}$, 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 $\mathsf{TTP} = \varnothing$, P_3 does the following:

- Assign $\mathbf{X}^\alpha_1 = \mathsf{Open}(\mathsf{pp}, c^{m_1^\alpha}_\alpha, o^{m_1^\alpha}_\alpha)$ and $\mathbf{X}^\alpha_{31} = \mathsf{Open}(\mathsf{pp}, c^{x_{31}^\alpha}_{2\ell+\alpha}, o^{x_{31}^\alpha}_{2\ell+\alpha})$ for $\alpha \in [\ell]$. Broadcast $(\texttt{conflict}, P_1)$ if Open results in $\perp$
- Assign $\mathbf{X}^\alpha_2 = \mathsf{Open}(\mathsf{pp}, c^{m_2^\alpha}_{\ell+\alpha}, o^{m_2^\alpha}_{\ell+\alpha})$, $\mathbf{X}^\alpha_{32} = \mathsf{Open}(\mathsf{pp}, c^{x_{32}^\alpha}_{3\ell+\alpha}, o^{x_{32}^\alpha}_{3\ell+\alpha})$ for $\alpha \in [\ell]$. Then broadcast $(\texttt{conflict}, P_2)$ if Open results in $\perp$
- Else, set $\mathbf{X} = \mathbf{X}_1 | \mathbf{X}_2 | \mathbf{X}_{31} | \mathbf{X}_{32}$, run $\mathbf{Y} \leftarrow \mathsf{Ev}(\mathbf{C}, \mathbf{X})$ and $y \leftarrow \mathsf{sDe}(\mathbf{Y})$. Broadcast $\mathbf{Y}$.

If P_3 broadcasts $(\texttt{conflict}, P_i)$, set $\mathsf{TTP} = P_{[2]\setminus i}$. If $\mathsf{TTP} = \varnothing$ and P_3 broadcasts $\mathbf{Y}$, P_i $(i \in [2])$ then do the following: Execute $y \leftarrow \mathsf{De}(\mathbf{Y}, d)$. If $y = \perp$, set $\mathsf{TTP} = P_1$.

Round 4: If $\mathsf{TTP} \neq \varnothing$: P_i $(i \in [2])$ sends x_i and o_{3i} (if valid) to TTP. P_3 sends o_{31}, o_{32} to TTP.

Round 5: TTP computes $x_{3i} = \mathsf{Open}(c_{3i}, o_{3i})$ 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 6.6: Protocol $\mathsf{g3PC}$

6.5 Transition from 5PC to 3PC

For better understanding, we describe how the transition from 5PC to 3PC takes place with a diagram when a conflict is identified and a 3PC instance is chosen. In such a case, input consistency must be maintained for 1) an x_{ij} that is held by the two garblers. 2) an x_{ij} that is held by one garbler, say P_α and evaluator P_γ. The case when all the three parties hold x_{ij} is subsumed in one of the above cases. The most critical case when x_{ij} is with only one of $\{P_\alpha, P_\beta, P_\gamma\}$ which is further categorized into two cases depending on whether 3) the input share is held either only by the garbler or 4) the input share is held by the evaluator. For the purpose of our explanation, we consider the case when a corrupt P_5 does not broadcast

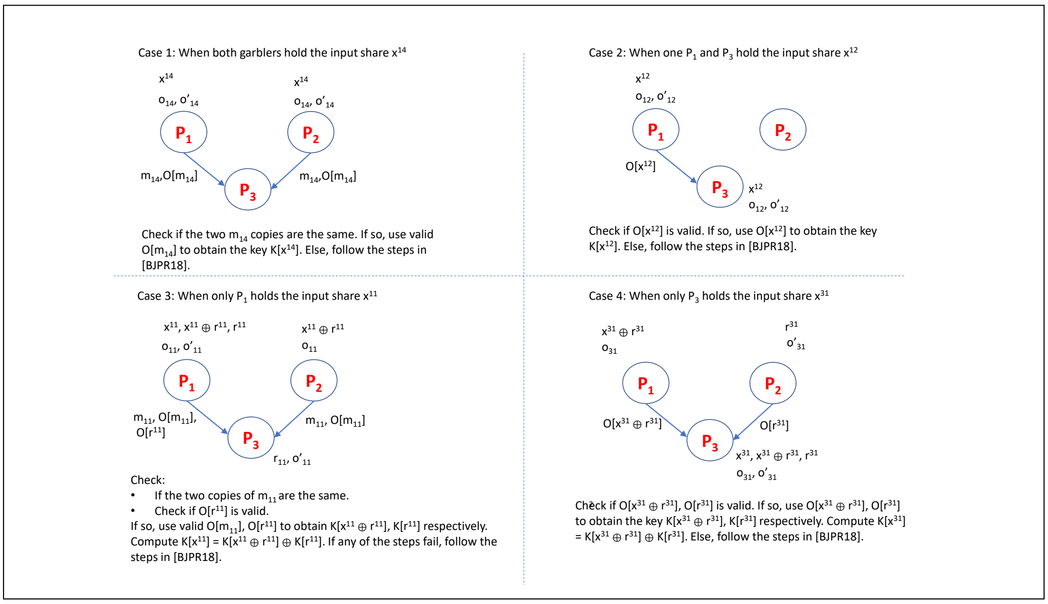

Figure 6.7: Diagram showing the transition from 5PC to 3PC.

Y and the garblers choose P_1, P_2, P_3 to run the robust 3PC of [BJPR18]. Hence, we have $\alpha = 1, \beta = 2, \gamma = 3$. We specifically consider the input shares of input x_1 of P_1 to describe the first 3 cases. We use the share of x_3 to describe case 4). For input x_1, P_1 holds all the shares (dealer), while P_2 holds (x^{14}, x^{15}, x^{16}) and P_3 holds (x^{12}, x^{13}, x^{16}). For input x_3, P_3 holds all the shares (dealer) while P_1 holds (x^{34}, x^{35}, x^{36}) and P_2 holds (x^{32}, x^{33}, x^{36}).

In the Fig 6.7, p_{ij} denotes the permutation bit for input x^{ij} and thus the commitments on both input keys for wire belonging to x^{ij} are sent in permuted order as per p_{ij}. m_{ij} denotes the XOR of x^{ij} and p_{ij}. Recall that as per $\mathsf{inputGOD}_i$, $(\mathsf{c}_{ij}, \mathsf{o}_{ij})$ denotes the commitment-opening pair for share $x_{ij} \oplus r_{ij}$ while $(\mathsf{c}'_{ij}, \mathsf{o}'_{ij})$ denotes the commitment-opening pair for share r_{ij} and all the commitments are broadcast, while the openings are sent privately. During the transition from 5PC to 3PC, for the shares of the form say x^{11} that are held by only one party, P_1 in the 3PC (the other two share holders are eliminated), the opening o_{11} (for share $x_{11} \oplus r_{11}$) is distributed to say P_2 while the opening o'_{11} (for share r_{11}) is distributed to P_3. Similar steps are done for the lone input share x^{31} held by P_3 and all others held by only one party in 3PC.

Inside the 3PC instance, in case 1) x^{14} is held by both garblers and not by the evaluator P_3. The garblers broadcast m_{14} and send the opening $\mathsf{O}[m_{14}]$ corresponding to the key $\mathsf{K}[x^{14}]$. If the copies of m_{14} match, then P_3 uses a valid opening $\mathsf{O}[m_{14}]$ (one of the two sent by the garblers) to get the key $\mathsf{K}[x^{14}]$. Else, the conflict resolution steps in [BJPR18] are followed. In case 2), x^{12} is held by garbler P_1 and evaluator P_3. The garbler P_1 sends $\mathsf{O}[x^{12}]$ to P_3 who checks if $\mathsf{O}[x^{12}]$ is valid. If so, P_3 uses opening $\mathsf{O}[x^{12}]$ to get the key $\mathsf{K}[x^{12}]$. Else, the conflict resolution steps in [BJPR18] are followed. In case 3), x^{11} is held only by garbler P_1. However the re-shares $x^{11} \oplus r^{11}$ and r^{11} are held respectively by P_2, P_3 (which are both known to P_1 due to $\mathsf{inputGOD}_1$). Now, P_1 sends m_{11} (masked bit wrt share $x^{11} \oplus r^{11}$) and $\mathsf{O}[m_{11}], \mathsf{O}[r^{11}]$ to P_3, while P_2 sends m_{11} and $\mathsf{O}[m_{11}]$ to P_3. P_3 now verifies if: the copies of m_{11} sent by the garblers are the same, the opening $\mathsf{O}[r^{11}]$ sent by P_1 is valid. If so, P_3 obtains the keys $\mathsf{K}[x^{11} \oplus r^{11}]$ and $\mathsf{K}[r^{11}]$ from the openings and XORs them to get $\mathsf{K}[x^{11}]$. If any of the checks fail, the conflict resolution steps in [BJPR18] are followed. In Case 4), where the evaluator alone holds the share x^{31} is simpler than case 3). However, the re-shares $x^{31} \oplus r^{31}$ and r^{31} are held respectively by P_1, P_2 (which are both known to P_3 due to $\mathsf{inputGOD}_3$). Now, P_1 sends $\mathsf{O}[x^{31} \oplus r^{31}]$ to P_3, while P_2 sends $\mathsf{O}[r^{31}]$ to P_3. P_3 now verifies if the openings are valid. If so, P_3 obtains the keys $\mathsf{K}[x^{31} \oplus r^{31}]$ and $\mathsf{K}[r^{31}]$ from the openings and XORs them to get $\mathsf{K}[x^{31}]$.

Every input share belongs to one of the above described four cases and is handled in a similar way. If all the input keys are obtained, P_3 evaluates the Yao's GC constructed by the garblers as per [BJPR18] and distributes the output to the garblers. Finally, the 3PC communicates the output to all the parties in 5PC. This completes the description.

6.6 Security Proof of god5PC

In this section, we outline the complete security proof of Theorem 6.3.5 that describes the security of our god5PC protocol relative to its ideal functionality in the standard security model.

Proof. We describe the simulator $\mathcal{S}_{\mathsf{god5PC}}$ for two cases which exhaustively cover the corruption scenarios: First, when P_1 and P_2 are corrupt. Second, when P_1 and P_5 are corrupt. The corruption of any two garblers is symmetric to the case when P_1, P_2 are corrupt and the corruption of any one garbler and evaluator is symmetric to the case of P_1, P_5 corrupt. The simulator acts on behalf of all honest parties in the execution. For better understanding we separate out the simulation for the subroutine inputGOD from the simulation of main protocol. In the inputGOD routine, we outline the simulator for the case of corrupt P_1, P_2 describing $\mathsf{inputGOD}_1$ for P_1's input x_1 and $\mathsf{inputGOD}_3$ for honest party's input x_3. The simulation of inputGOD routine for the case of corrupt P_1, P_5 is identical to the case of corrupt P_1, P_2. The inputs of corrupt parties are extracted in the inputGOD routine.

We give a high level view of the simulation of garbling and output computation as follows: First, in the case of P_1^*, P_2^* corrupt, the evaluator P_5 is honest. Hence, in this case, correctness is required from the distributed GC. The simulator behaves as an honest $P_i, i \in \{3, 4\}$ by raising conflicts as per the protocol in case of any cheating throughout the garbling phase, since all seeds are known to the adversary. If no cheating is detected throughout the GC construction, then a GC is generated as per the Garble_4 procedure. Else a 3PC instance is identified and the simulator in turn invokes the simulator of 3PC guaranteed output delivery protocol to complete the simulation. Second, in the case of P_1^*, P_5^* corrupt, the simulator knows the seeds held by the adversary. In addition the simulator has complete control over the part of GC generated using the seed s_2. Since input extraction is done in the $\mathsf{inputGOD}$ routine, the simulator can invoke the functionality to obtain y in advance at the time of garbling. As a result with the knowledge of y, a fake garbled circuit is constructed by the simulator using s_2 that always evaluates to the same output keys forming the output super-key $\mathbf{Y}$, which correspond to the evaluation performed using the extracted inputs of the adversary and the inputs of the honest parties. The output masking bit share λ_w^2 for each output wire w is broadcasted after setting it to $(y \oplus (\oplus_{i \in [4], i \neq 2} \lambda_w^i))$ in the garbling phase itself since the simulator knows y and all masking bit shares in advance. Finally, if $\mathbf{Y}$ is received from P_5^* on behalf of honest parties then the simulation terminates, else a 3PC instance is identified according to the protocol and the simulator runs the simulator of the 3PC instance sub-routine to complete the simulation. (Since the simulator for 3PC is already well-described in [BJPR18], we do not provide details of it).

We describe the simulator steps in detail for $\mathsf{inputGOD}()$ and the main protocol separately in Figs 6.8 and 6.9, 6.10 respectively.

Simulator $\mathcal{S}^{12}_{\mathsf{inputGOD}_1}$

$\mathcal{S}^{12}_{\mathsf{inputGOD}_1}$ (for input x_1)

R1 Receive the broadcast commitments $\{\mathsf{pp}_1, \mathsf{c}_{1j}, \mathsf{c}'_{1j})\}_{j \in 6}$ on behalf of each $P_l, l \in \{3,4,5\}$ and openings $\{\mathsf{o}_{1j}, \mathsf{o}'_{1j}\}$ from P_1^* on behalf of $P_l, l \in \{3,4,5\}, P_l \notin \mathcal{T}_j$. For opening o_{13} corresponding to share x^{13} that is common between P_3, P_4, accept a default value if o_{13} sent by P_1^* and received on behalf of P_3 and P_4 are both invalid i.e., $\mathsf{Open}(\mathsf{pp}_1, \mathsf{c}_{13}, \mathsf{o}_{13}) = \perp$. Else, accept the opening whichever is valid. Similar steps are done for openings o'_{13} and for shares common between P_3, P_5 and P_4, P_5 as well.

R2 Send openings corresponding to commitments $\mathsf{c}_{16}, \mathsf{c}_{15}, \mathsf{c}_{14}$ on behalf of P_3, P_4, P_5 respectively to P_2^*. Similarly, receive openings $\mathsf{o}_{16}, \mathsf{o}_{15}, \mathsf{o}_{14}$ on behalf of P_3, P_4, P_5 respectively from P_2^*. For opening o_{16} of share x^{16} that is common between P_2^*, P_3, accept a default value if o_{16} received on behalf of P_3 from P_1^* and sent by P_2^* are both invalid. Else, accept the opening received from either P_1^*, P_2^* whichever is valid. Similar steps are done for opening o'_{16} common between P_2^*, P_3 and openings common between P_2^*, P_4 $(\mathsf{o}_{15}, \mathsf{o}'_{15})$ and P_2^*, P_5 $(\mathsf{o}_{14}, \mathsf{o}'_{14})$. Compute $x_1 = \oplus_{j \in [6]} x^{ij}$.

$\mathcal{S}^{12}_{\mathsf{inputGOD}_3}$ (for input x_3)

R1 On behalf of P_3: Compute $\{\mathsf{pp}_3, \mathsf{c}_{3j}, \mathsf{c}'_{3j}\}$ as commitments on randomly chosen x^{3j}, r^{3j} for $j \in [6]$ such that for $l \in [2]$ it holds that $P_l^* \notin \mathcal{T}_j$. For remaining shares such that $P_l^* \in \mathcal{T}_j$, compute commitments on dummy value. Broadcast $\{\mathsf{c}_{3j}, \mathsf{c}'_{3j}\}_{j \in 6}$ on behalf of P_3 and send openings $\{\mathsf{o}_{3j}, \mathsf{o}'_{3j}\}_{j \in 6, P_l^* \notin \mathcal{T}_j}$ to P_l^*.

R2 Send openings $\mathsf{o}_{35}, \mathsf{o}_{34}$ (corresponding to commitments $\mathsf{c}_{35}, \mathsf{c}_{34}$) to P_1^* and $\mathsf{o}_{33}, \mathsf{o}_{32}$ (corresponding to $\mathsf{c}_{33}, \mathsf{c}_{32}$) to P_2^* on behalf of P_4 and P_5 respectively. Similar steps are done for openings $\mathsf{o}'_{35}, \mathsf{o}'_{34}$ common between P_1^*, P_4 and $\mathsf{o}'_{33}, \mathsf{o}'_{32}$ common between P_2^*, P_5.

Figure 6.8: Simulator $\mathcal{S}^{12}_{\mathsf{inputGOD}_1}$ (for input x_1) with actively corrupt P_1^*, P_2^*

Simulator $\mathcal{S}^{12}_{\mathsf{god5PC}}$

$\mathcal{S}^{12}_{\mathsf{god5PC}}$ (P_1^*, P_2^* are corrupt)

Input and Seed Distribution Phase.

– Simulation of $\mathcal{S}^{12}_{\mathsf{inputGOD}_i}, i \in [5]$ instances for input x_i. Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_1), (Input, x_2) on behalf of P_1^*, P_2^* to obtain y.

– For simulation of $\mathsf{seedGOD}_g, g \in [2]$, receive $(\mathsf{pp}^g, \mathsf{c}_g)$ from P_g^* on behalf of all honest parties. Receive o_g on behalf of P_3 and P_4 from P_g^*. If a valid opening o_g is received on behalf of at least one of P_3, P_4, use the corresponding valid opening to obtain s_g. Else assume a default value for s_g.

– For simulation of $\mathsf{seedGOD}_g, g \in \{3, 4\}$, sample random s_g and compute $(\mathsf{c}_g, \mathsf{o}_g) \leftarrow \mathsf{Com}(\mathsf{pp}^g, \mathsf{s}_g)$. Broadcast $(\mathsf{pp}^g, \mathsf{c}_g)$ on behalf of P_g and send o_g on behalf of P_g to P_1^*, P_2^*.

Garbling Phase.

– For simulation of Round 1 of Garble_4, it is necessary to ensure correctness of the circuit. Behave as honest $P_g, g \in \{3, 4\}$ using the seeds chosen in Round 1. Simulate each instance of Π_{4AOTGOD} by acting as an honest party. If a Π_{4AOTGOD} instance returns $\mathcal{P}^3$ (due to inconsistent messages from either P_1^* or P_2^*), invoke $\mathcal{S}_{\mathsf{god3PC}}$ (Simulator for 3PC [BJPR18]) and send the output y to all received from the simulation of $\mathsf{god3PC}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

– For simulation of Round 2 of Garble_4, behave as honest $P_g, g \in \{3, 4\}$. If a Π_{4AOTGOD} instance returns $\mathcal{P}^3$ (due to inconsistent messages from either P_1^* or P_2^*) or $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$ is identified when (P_α, P_β) with $\alpha, \beta \in \mathcal{S}_j$ for some $j \in [4]$ broadcasts different GC^j, invoke $\mathcal{S}_{\mathsf{god3PC}}$ and send the output y to all received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation. If there is no conflict in the garbling phase, then the GC (described in Fig.3.5) will be the output of honest parties.

Masked input bit and Key Transfer Phase.

– For $i \in \{3,4\}$ and $j \in \mathcal{S}_i \backslash \mathcal{S}_g$, do as per the protocol: broadcast λ_w^j for each input wire w belonging to P_g^* where $g \in [2]$ and λ_w^l for each output wire w on behalf of P_i where $l \in \mathcal{S}_i$. Broadcast λ_w^β on behalf of honest P_i for input wire w belonging to honest $P_{g'}$ where $g' \in \{3,4\} \setminus \{i\}$ and $g' \notin \mathcal{S}_i$. Also, receive on behalf of the honest P_i, λ_w^α (for each input wire w) where $\alpha \notin \mathcal{S}_i$ and λ_w^l (for each output wire w) from $P_g^*, g \in [2]$ where $l \in \mathcal{S}_g$. If for any α, l, the received $\lambda_w^\alpha / \lambda_w^l$ from P_g^*, does not correspond to the one generated using s_g, then invoke $\mathcal{S}_{\mathsf{god3PC}}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_g^*, P_\beta\}$, where $\beta \in \mathcal{S}_g$ is the index of the party in conflict with P_g^* and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

– For each wire w corresponding to input $x_w = x^{ij}$ held by $P_\alpha^*, \alpha \in [2] \cap \mathcal{X}_{ij}$, compute the masked input $b_w = x_w \oplus \lambda_w$ as per the protocol and broadcast b_w on behalf of $P_l, l \in (\{3,4\} \cap \mathcal{X}_{ij})$. Also receive b_w from P_α^* on behalf of honest parties. If the received b_w for any w from P_α^* does not match with the one originally broadcasted by P_l, then invoke $\mathcal{S}_{\mathsf{god3PC}}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha^*, P_l\}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

- For each wire w holding the input share $x_w = x^{ij}$ belonging to only honest parties, broadcast random b_w on behalf of the honest parties.

– For every input wire w, where $\{k_{w,0}^g, k_{w,1}^g\}_{g \in [4]}$ denote the super-key derived from seeds $\{\mathsf{s}_g\}_{g \in [4]}$, each $P_l, l \in \{3,4\}$ computes commitments on these as per the protocol steps and broadcasts $\{c_{w,b}^j\}_{b \in \{0,1\}, j \in \mathcal{S}_l}$ on behalf of P_l. Also receive on behalf of the honest parties, $\{c_{w,b}^j\}_{b \in \{0,1\}}$ sent by $P_\alpha^*, \alpha \in [2] \cap \mathcal{S}_l$. If the commitment received for any w from P_α^* does not match with the one originally created on behalf of P_l, then invoke $\mathsf{god3PC}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_\alpha^*, P_l\}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

Evaluation and Output Phase.

– Compute $\mathbf{Y}$ such that for all output wires w, each key in $\mathbf{Y}$ maps to $(y_w \oplus \lambda_w)$. Broadcast $\mathbf{Y}$ on behalf of P_5.

Figure 6.9: Simulator $\mathcal{S}_{\mathsf{god5PC}}^{12}$ for $\mathsf{god5PC}$ with actively corrupt P_1^*, P_2^*

The hybrid arguments are as follows:

Security against corrupt P_1^*, P_2^*: We now argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{god5PC}}^{12}} \overset{c}{\approx} \textsc{real}_{\mathsf{god5PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1, P_2. The views are shown to be indistinguishable via a series of intermediate hybrids.

– $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{god5PC}, \mathcal{A}}$.

– $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except that when the execution does not result in P_1^*, P_2^* getting access to the opening of the commitment $\mathsf{c}_{ij}, i \in \{3,4,5\}, j \in [6]$ in the $\mathsf{inputGOD}_i$, the commitment is replaced with the commitment of a dummy value.

– HYB_2: Same as HYB_1 except that P_5 raises a conflict to identify a 3PC instance if any decommitment for $\{k^g_{w,0}, k^g_{w,1}\}_{g\in[4]}$ corresponding to a committed share not held by P_5 opens to a value other than what was originally committed and held by $P^*_i, i \in [2]$.

– HYB_3: Same as HYB_3 except that $\mathbf{Y}$ is computed as $\mathbf{Y} = \{k^g_{w,y_w\oplus\lambda_w}\}_{g\in[4]}$ for each output wire w instead of running the Evaluation Phase of garbling.

– HYB_4: Same as HYB_3 except that in case of a 3PC instance elected, run $\mathcal{S}_{\mathsf{god3PC}}$ in place of the 3PC protocol algorithm.

Note that $\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^{12}_{\mathsf{god5PC}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference between the hybrids is that in HYB_1, when the execution does not result in P^*_1, P^*_2 getting access to the opening of commitments $\mathsf{c}_{ij}, i \in \mathcal{P}_{12}, j \in [6]$ in the $\mathsf{inputGOD}_i$, the commitment is replaced with the commitment of a dummy value. The indistinguishability follows from the hiding property of the commitment scheme.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that in HYB_1, P_5 raises a conflict if the decommitment for $\{k^g_{w,0}, k^g_{w,1}\}_{g\in[4]}$ corresponding to a committed share not held by P_5 and sent by $P^*_i, i \in [2]$ is invalid (the decommitment is $\perp$) whereas in HYB_2, P_5 raises a conflict to identify the 3PC instance if the decommitment corresponding a committed share opens to a value other than what was originally committed and held by P^*_i. Since the commitment scheme Com is binding for any pp, P^*_i could have successfully decommitted to a value than what was originally committed with negligible probability. Hence, the hybrids are indistinguishable.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The only difference between the hybrids is that, in HYB_3, $\mathbf{Y}$ is computed as $\mathbf{Y} = \{k^g_{w,y_w\oplus\lambda_w}\}_{g\in[4]}$ instead of running the Evaluation Phase of the garbling. The indistinguishability follows from the correctness of the garbling scheme since $\mathbf{Y}$ computed using the Evaluation Phase of garbling would also result in $\mathbf{Y} = \{k^g_{w,y_w\oplus\lambda_w}\}_{g\in[4]}$

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The only difference between the hybrids is that, in HYB_3, a real-world 3PC is run in case of conflict whereas $\mathcal{S}_{\mathsf{god3PC}}$ is run in HYB_4. Since, $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}_{\mathsf{god3PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{god3PC},\mathcal{A}}$ [BJPR18], indistinguishability follows.

Simulator $\mathcal{S}^{15}_{\mathsf{god5PC}}$

$\mathcal{S}^{15}_{\mathsf{god5PC}}$ (P_1^*, P_5^* are corrupt)

Input and Seed Distribution Phase.

– Simulation of $\mathcal{S}^{15}_{\mathsf{inputGOD}_i}, i \in [5]$ instances for input x_i. Invoke $\mathcal{F}_{\mathsf{god}}$ with (Input, x_1), (Input, x_5) on behalf of P_1^*, P_5^* to obtain y.

– For simulation of $\mathsf{seedGOD}_1$, receive $(\mathsf{pp}^1, \mathsf{c}_1)$ from P_1^* on behalf of all honest parties. Receive o_1 on behalf of P_3 and P_4 from P_1^*. If there exists a valid opening o_1 received on behalf of at least one of P_3, P_4, use the corresponding valid opening to obtain s_1. Else assume a default value for s_1.

– For simulation of $\mathsf{seedGOD}_g, g \in \{3, 4\}$, sample random s_g and compute $(\mathsf{c}_g, \mathsf{o}_g) \leftarrow \mathsf{Com}(\mathsf{pp}^g, \mathsf{s}_g)$. Broadcast $(\mathsf{pp}^g, \mathsf{c}_g)$ on behalf of P_g and send o_g on behalf of P_g to P_1^*. For $\mathsf{seedGOD}_2$, broadcast random commitment $(\mathsf{pp}^2, \mathsf{c}_2)$ on behalf of P_2.

Garbling Phase.

– For simulation of Round 1 of Garble_4 on behalf of honest $P_l, l \in \{2, 3, 4\}$, all the seeds are known. Additionally, s_2 is not known to P_1^*, so the randomness and GC^2 generated using s_2 is unknown to P_1^*. Use the y obtained from the $\mathcal{F}_{\mathsf{god}}$ to compute $\lambda^2_w = y \oplus \lambda^1_w \oplus \lambda^3_w \oplus \lambda^4_w$ for each output wire w. Participate in the distributed garbling as before but constructing a simulated GC with the help of s_2 and with the knowledge of y such that each ciphertext encrypts the same output key that represents the masked output which corresponds to the evaluation performed using the extracted inputs of the adversary and the inputs of the honest parties. Simulate each instance of Π_{4AOTGOD} by acting as honest party. If a Π_{4AOTGOD} instance returns $\mathcal{P}^3$ (due to inconsistent messages from P_1^*), invoke $\mathcal{S}_{\mathsf{god3PC}}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

– For simulation of Round 2 of Garble_4, compute the simulated garble circuit using s_2 on behalf of $P_l, l \in \{2, 3, 4\}$. If a Π_{4AOTGOD} instance returns $\mathcal{P}^3$ (due to inconsistent messages from P_1^*) or $\mathcal{P}^3 = \mathcal{P} \setminus \{P_1^*, P_\beta\}$ is identified when (P_1^*, P_β) with $1, \beta \in \mathcal{S}_j$ for some $j \in [4]$ broadcasts different GC^j , invoke $\mathcal{S}_{\mathsf{god3PC}}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation. If there is no conflict in the garbling phase, then the GC (described in Fig.3.5) will be the output of honest parties.

Masked input bit and Key Transfer Phase.

– For $i \in \{2, 3, 4\}$ and $j \in \mathcal{S}_i$, do as per the protocol: broadcast λ^j_w for each input wire w belonging to P_5^*. For $j \notin \mathcal{S}_1$, broadcast λ^j_w for each input wire w belonging to P_1^* and λ^l_w (for each output wire w) on behalf of P_i where $l \in \mathcal{S}_i$. Broadcast λ^β_w on behalf of honest P_i for input wire w belonging to honest $P_{g'}$ where $g' \in \{2, 3, 4\} \setminus \{i\}$ and $\beta \notin \mathcal{S}_g$. Also, receive on behalf of the honest P_i, λ^α_w (for each input wire w) where $\alpha \notin \mathcal{S}_i$ and λ^l_w (for each output wire w) from P_1^* where

$l \in \mathcal{S}_1$. If for any α, l, the received $\lambda_w^\alpha / \lambda_w^l$ from P_1^*, does not correspond to the one generated on behalf of the honest parties, then invoke $\mathcal{S}_{\mathsf{god3PC}}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_g*, P_\beta\}$, with $\beta \in \mathcal{S}_g$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

- For each wire w corresponding to input $x_w = x^{ij}$ held by P_1^* and two honest garblers, set the masked input $b_w = x_w \oplus \lambda_w$ as per the protocol and broadcast b_w on behalf of $P_l, l \in (\{2,3,4\} \cap \mathcal{X}_{ij})$. Also receive b_w from P_1^* on behalf of honest parties. Also, for x_w held by only honest parties, broadcast a random b_w on behalf of all honest parties. If the b_w received for any w from P_1^* does not match with the one created on behalf of honest P_l, then invoke $\mathcal{S}_{\mathsf{god3PC}}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_1^*, P_l\}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.
- For every input wire w, where $\{k_{w,0}^g, k_{w,1}^g\}_{g \in [4]}$ denote the super-keys derived from seeds $\{\mathsf{s}_g\}_{g \in [4]}$, on behalf of each $P_l, l \in \{3,4\}$ compute commitments on these as per the protocol steps for all seeds except s_2. For commitments in $(c_{w,0}^j, c_{w,1}^j)$ obtained using s_2 that correspond to input keys, generate commitments to the shares as per NICOM. Commit to dummy values for all other keys that are not input keys. Broadcast $\{c_{w,b}^i\}_{b \in \{0,1\}, i \in \mathcal{S}_\alpha}$ on behalf of $P_\alpha, \alpha \in \{2,3,4\}$. Also receive $\{c_{w,b}^j\}_{b \in \{0,1\}}$ sent by $P_1^*, j \in \mathcal{S}_1$ on behalf of the honest parties. If the commitment received for any w from P_1^* does not match with the one originally created on behalf of honest P_β, where $\beta \in \mathcal{S}_1$, then invoke $\mathcal{S}_{\mathsf{god3PC}}$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_1^*, P_\beta\}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

Evaluation and Output Phase.

- Receive $\mathbf{Y}$ from P_5^* on behalf of $P_g, g \in \{2,3,4\}$. If received $\mathbf{Y}$ for some output wire w and index $j \in \mathcal{S}_g$ does not match with the output super-key created in the generation of simulated GC, invoke $\mathcal{S}_{\mathsf{god3PC}}$ with $\mathcal{P}^3$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_1^*, P_5^*\}$ and send the output y received from $\mathcal{S}_{\mathsf{god3PC}}$ on behalf of honest parties in $\mathcal{P}^3$ to complete the simulation.

Figure 6.10: Simulator $\mathcal{S}_{\mathsf{god5PC}}^{15}$ for $\mathsf{god5PC}$ with actively corrupt P_1^*, P_5^*

Security against corrupt P_1^*, P_5^*: We now argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{god5PC}}^{15}} \stackrel{c}{\approx} \textsc{real}_{\mathsf{god5PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1, P_5. The views are shown to be indistinguishable via a series of intermediate hybrids.

- $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{god5PC}, \mathcal{A}}$.
- $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except that when the execution does not result in P_1^*, P_5^* getting access to the opening of the commitment $\mathsf{c}_{ij}, i \in \{2,3,4\}, j \in [6]$ in $\mathsf{inputGOD}_i$, the commitment is replaced with the commitment of a dummy value.
- $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except that the commitment to seed s_2 in $\mathsf{seedGOD}_2$ is replaced with the commitment on dummy value.

- $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that some of the commitments of input keys sent by P_2, P_3, P_4 wrt seed s_2, which will not be opened are replaced with commitments of dummy values. These commitments correspond to the labels that do not correspond to any input share.

- $\textsc{hyb}_4$: Same as $\textsc{hyb}_3$ except that the GC is created as simulated one with the knowledge of s_2 and output y along with the share λ_w^2 for each output wire w set to the value $\lambda_w^2 = y \oplus (\oplus_{i \in [4], i \neq 2} \lambda_w^i)$.

- $\textsc{hyb}_5$: Same as $\textsc{hyb}_4$ except that a 3PC instance is chosen as per the protocol if the received $\mathbf{Y}$ does not correspond to the $\mathbf{Y}$ originally created by the simulated GC. Note that $\textsc{hyb}_5 = \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^{15}_{\mathsf{god5PC}}}$.

- $\textsc{hyb}_6$: Same as $\textsc{hyb}_5$ except that in case of a 3PC instance elected, run $\mathcal{S}_{\mathsf{god3PC}}$ in place of the 3PC protocol algorithm.

Next, we show that each pair of hybrids are computationally indistinguishable as follows:

$\textsc{hyb}_0 \overset{c}{\approx} \textsc{hyb}_1$: The only difference between the hybrids is that, in $\textsc{hyb}_1$, when the execution does not result in P_1^*, P_5^* getting access to the opening of commitments $\mathsf{c}_{ij}, i \in \{2, 3, 4\}, j \in [6]$ in the $\mathsf{inputGOD}_i$, the commitment is replaced with the commitment of a dummy value. The indistinguishability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_1 \overset{c}{\approx} \textsc{hyb}_2$: The only difference between the hybrids is that, in $\textsc{hyb}_2$, the commitment to the seed s_2 is replaced with the commitment on a dummy value. The indistinguishability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_3$: The only difference between the hybrids is that, in $\textsc{hyb}_3$, the commitments of input wire labels wrt seed s_2, which will not be opened are replaced with commitments on dummy values. The indistingushability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_3 \overset{c}{\approx} \textsc{hyb}_4$: The only difference between the hybrids is that in $\textsc{hyb}_4$, GC is constructed as a simulated one using the seed s_2 and the knowledge of output y instead of a real GC. More concretely, In $\textsc{hyb}_3$, Rounds 1, 2 are run as per Garble_4, which gives GC. In $\textsc{hyb}_4$, it is generated as a simulated circuit and additionally, for each output wire w, λ_w^2 is set to $\lambda_w^2 = y \oplus (\oplus_{i \in [4], i \neq 2} \lambda_w^i)$. Indistinguishability follows from reduction to the security of distributed garbling which in turns relies on the the double-keyed PRF F.

$\text{HYB}_4 \stackrel{c}{\approx} \text{HYB}_5$: The only difference between the hybrids is that, in HYB_4, a 3PC instance is identified if k^j_{w,b_w} of the received $\mathbf{Y}$ for some output wire w and index $j \in \mathcal{S}_g$ does not match with either $(k^j_{w,0}, k^j_{w,1})$ or the three keys $k^j_{w,b_w}, j \in \mathcal{S}_g$ in $\mathbf{Y}$ do not map to the same b_w whereas in HYB_5, a 3PC committee is identified if the received $\mathbf{Y}$ does not match the one created using simulated GC. By security of the garbling scheme, P_5 could have forged such a $\mathbf{Y}$ only with negligibility probability.

$\text{HYB}_5 \stackrel{c}{\approx} \text{HYB}_6$: The only difference between the hybrids is that, in HYB_5, a real-world 3PC is run in case of conflict whereas $\mathcal{S}_{\mathsf{god3PC}}$ is run in HYB_6. Since, $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{god3PC}}} \stackrel{c}{\approx} \text{REAL}_{\mathsf{god3PC}, \mathcal{A}}$ [BJPR18], indistinguishability follows. $\square$

Part II

Four-Party Computation with Mixed Adversary

Chapter 7

4PC with GOD

In this section, we present an efficient constant round 4PC protocol achieving the strongest security notion of GOD against an adversary in mixed model who corrupts 2 parties such that one is active and the other is passive ($t_a = 1$, $t_p = 1$). We rely only on pairwise private channels for communication. This protocol is yet again inspired from [CGMV17] that promises selective abort for 5 parties against 2 active corruptions (honest majority). We customize their techniques to achieve a much stronger notion of GOD in our setting which is even stronger than strict honest majority. To provide robustness, similar to god5PC, we ensure public identification of conflict between two parties (one of which is surely actively corrupt) in case of *any* adversarial mis-behaviour and switch to a passive 2PC based on Yao's garbled circuit [Yao82] to obtain the output.

7.1 The Construction

The protocol retains 3 parties $\{P_1, P_2, P_3\}$ as garblers and P_4 as evaluator. Our protocol can be segregated into phases, some of which can be run in parallel to minimize rounds. At a high level, we begin with seed and input commit phase, then run the garbling phase that involves some non-trivial techniques for the transfer of keys for input wires and conclude with the evaluation phase and computation of output. For garbling, we use a one-time seed-distribution (SD) as in π_{seedDist} (Fig 3.7) where the seeds $\{\mathsf{s}_1, \mathsf{s}_2, \mathsf{s}_3\}$ are distributed amongst the garblers $\{P_1, P_2, P_3\}$ s.t a garbler P_g knows all but seed s_g. The key feature of our construction is the use of only semi-honest primitives, namely *distributed garbling* and *oblivious transfer* [EGL85], despite having a malicious party out of the two corruptions (dishonest majority). For smoother description, we first describe the protocol assuming an additional broadcast channel and later realize each broadcast with EIG protocol [BNDDS87] with threshold $n > 3t_a$ in 3 rounds.

The key idea to ensure GOD is to employ tools to eliminate the sole actively-corrupt party in case of any wrongdoing and further rely on the remaining parties to robustly compute the output. However, the techniques used on top of the passively secure primitives to provide security against the active corruption are not always sufficient to pin-point the malicious party. Thus, in case the adversary strikes, we resort to unanimously identifying a conflict between two parties, one of which is guaranteed to be the actively corrupt and eliminate them. The remaining two parties can run a 2PC [Yao82], which is robust for one semi-honest corruption. As a result, achieving *guaranteed output delivery* in the mixed model boils down to resolving the following two challenges: (a) unanimous identification and elimination of conflict in case of any misbehavior leading to abort; (b) ensuring input consistency across the 4PC and the smaller 2-party instance to prevent the adversary from obtaining outputs on multiple inputs.

Case (b) is particularly tricky when, the actively-corrupt evaluator sends an invalid output (or *no* output) to the garblers after learning the output herself on successful evaluation. We address this concern by having an input-commit phase where each party additively splits her input into 3 shares, distributes shares s.t each shareholder gets one share. We force the dealer to commit to her input via these shares (else a default is chosen) using the *commit publicly, open privately* technique where each party generates commitment on the shares, broadcasts the commitments and sends the opening of each share privately to exactly one party to provide resilience against 2 corruptions. Besides, input-privacy, this further ensures that the actively-corrupt party is bound to her input across the 4PC and 2PC runs. To elaborate, the parties that run 2PC possess all but one share of every input, which is held by both the eliminated parties. These eliminated parties are enabled to provide their share to exactly one party in the 2PC for further computation. Since one of the eliminated parties is honest or passive, the 2PC instance always receives at least one valid opening for that share, thus ensuring input consistency. However, note that releasing this share to a party in 2PC who does not possess it, still preserves input privacy since the share already belongs to the adversary.

Case (a) is dealt with based on whether the inconsistency was detected in (i) *broadcast resilient* data or (ii) *private* data. Case (i) may involve either input-independent data such as GC, mask-shares of output wires, blinded-input and mask-shares (wrt seeds not held by the wire owner) on input wires, all of which can be generated by *two* parties who share the same seed or input-share. On the similar lines as god5PC, correctness of such data can be determined by simply comparing the copies of broadcast data (one of the two senders is honest/passive). Also, broadcast of such data does not cause any privacy leaks. Since one of the two seed owners is honest or passive, any wrongdoing can be determined by simply comparing the copies of broadcasted message. Consequently, if a conflict is established, the conflicting parties are eliminated. However, case (ii) involves communication of input-dependent data such as keys used for evaluation where privacy is crucial. This is handled as explained below.

The transfer of keys on input wires involves sending of keys for each fragment of GC. To ensure that each input key indeed corresponds to the masked input share, each garbler commits to both the keys for every input wire of the DGC fragment generated by her as in [MRZ15, CGMV17]. The evaluator needs keys corresponding to all GC fragments for every input share to perform evaluation. The input commit phase ensures that, each input share is held by two parties. Hence for each input shares that are held by two garblers say P_i, P_j, P_i, P_j together are aware of all seeds and thus each sends openings for the commitments on input key corresponding to the seeds they own. Consequently, if any opening is invalid, the evaluator raises a public conflict with the sender and the two get eliminated while the remaining two parties run a 2PC.

A trickier case occurs for the transfer of input keys belonging to an input share held by a garbler (P_g) and the evaluator (P_4). P_g can send only 2 out of the 3 keys (for seeds in $\mathcal{S}_g$) as P_g does not possess s_g. Hence there is need for a way to communicate the input keys for the DGC fragment GC^g. We use passively secure 1-out-of-2 OTs to communicate the residual input key corresponding to s_g. At the first glance, although it appears that actively-secure OTs must be employed due to the presence of a malicious party, we use neat tricks to ensure *privacy* and *correctness* while relying on passive-OTs. It is observed that, the passively secure 1-out-of-2 of [EGL85] is already secure against a maliciously corrupt sender. For security against a malicious receiver, we employ techniques outside of semi-honest OT to protect the privacy of the sender. To elaborate, we split the sender's message (both keys of input wire) into two additive shares, generate commitments on them. Then we run two instances of semi-honest OT involving two different pairs of parties with sender in each OT holding openings for commitments on one additive share of both the input keys and the receiver in each OT holding the same choice-bit. For instance, for the input share x^{14} held by P_1, P_4, the seed s_1 is held by P_2, P_3 who split the openings for the keys belonging to x^{14}. P_1 acts as a receiver with her masked input of x^{14} as the choice bit and runs an OT with P_2 as a sender. Similarly, P_4 acts as receiver with the same choice bit as P_1 and runs an OT with P_3 as sender. Thus, if P_4 is maliciously corrupt and P_1 is passive, then only P_4 learns P_3's inputs for the OT which are random additive shares and P_1 learns nothing since the OT is secure against a passive receiver. Further, if one of the sender say P_3 is malicious, then the obtained opening may be invalid (leading to $\perp$) and thus P_4 will publicly raise a conflict that leads to P_1, P_2 running a 2PC instance. With this technique, we achieve our purpose while preserving correctness and privacy.

To ensure the robustness of 3- party garbling, we modify $\mathcal{F}_{\mathsf{3AOT}}$ to tackle the abort cases.

The modified AOT is presented in Fig 7.1.

Protocol Π_{3AOTGOD}

P_s, P_r denote the sender and receiver respectively. P_a denotes the attester and P_h denotes the auditor.

Input and Output: P_s inputs m_0, m_1, P_r inputs choice bit b. P_r outputs $m_b/\mathcal{F}$. P_a outputs $\perp/\mathcal{F}$.

Notations $\mathcal{F}$ denotes the set of two parties in conflict one of which is guaranteed to be actively corrupt.

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Chapter 2).

Round 1: P_s samples pp and random values $r_0, r_1 \leftarrow \{0,1\}^\kappa$ (derived from $\mathsf{s}_i, i \in \mathcal{S}_s \cap \mathcal{S}_a$) to compute $(\mathsf{c}_0, \mathsf{o}_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$ and $(\mathsf{c}_1, \mathsf{o}_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_s broadcasts $(\mathsf{pp}, \mathsf{c}_0, \mathsf{c}_1)$.

Round 2: P_a, who knows (r_0, r_1) (derived from s_i), also computes $(\mathsf{c}'_0, \mathsf{o}'_0) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)$ and $(\mathsf{c}'_1, \mathsf{o}'_1) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)$. P_a broadcasts $(\texttt{conflict}, P_s, P_a)$ and terminates the routine by setting $\mathcal{F} = \{P_s, P_a\}$ if $c_0 \neq c'_0$ or $c_1 \neq c'_1$. Else, it broadcasts $(\mathsf{c}'_0, \mathsf{c}'_1)$ and sends o'_b privately to P_r.

(**Computation by** P_r**:**) Set $\mathcal{F} = \{P_s, P_a\}$ if the values broadcast by P_s, P_a do not match. Broadcast $(\texttt{conflict}, P_s, P_r)$ and terminate by outputting $\mathcal{F} = \{P_r, P_a\}$ if no o_b is received or $\mathsf{Open}(\mathsf{c}_b, \mathsf{o}_b) = \perp$. Else, output $m_b = \mathsf{Open}(\mathsf{c}'_b, \mathsf{o}'_b)$.

(**Computation by** P_a, P_h, P_s**:**) If the values broadcast by P_s, P_a mismatch or got a conflict message, output $\mathcal{F}$. Else, output $\perp$.

Figure 7.1: Protocol $\Pi_{\mathsf{3AOTGOD}}(P_s, P_r, P_a, P_h)$ for $\mathsf{god4PC}$

We use extractable commitments to commit on the seeds in the seed-commit phase owing to a technicality arising in the proof. Elaborate details are presented in Section 7.3. It is interesting to note that, broadcast in our setting can be efficiently realized using any broadcast protocol with threshold $n > 3t_a$. We instantiate our broadcast with EIG broadcast [BNDDS87] of 3 rounds and eliminate the need of broadcast channel. Our seed distribution and passively secure 2PC protocol appear in Figs 3.7 and 7.2 respectively. The main protocol appears in Fig 7.3 and is explained with broadcast for simplicity.

Protocol $\mathsf{passive2PC}$

Notation: Let P_α and P_β be the two parties appointed to run 2PC. Let $\{P_m, P_n\} = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$ be the eliminated parties.

Inputs: P_α (similarly P_β) has $\{x^{\alpha i}\}_{i \in [4] \setminus \{\alpha\}}, x^{m\alpha}, x^{n\alpha}, x^{mn}$.

Output: Each party outputs $y = f(x_1, x_2, x_3, x_4)$.
Common Inputs: The circuit C that takes the additive shares x^{ij} of x_i for $i \in [4], j \in [4] \setminus \{i\}$ as inputs and computes $f(x_1, x_2, x_3, x_4)$, each input, their shares and output are from $\{0, 1\}$ (instead of $\{0, 1\}^\ell$ for simplicity).
Input distribution: P_m and P_n send the openings for commitment to input share x^{mn} i.e. o_{mn} to P_α who uses the valid opening (out of the two) to compute $x^{mn} \leftarrow \mathsf{Open}(\mathsf{pp}_i, c_{mn}, o_{mn})$. Similarly, P_m and P_n provide o_{nm} to P_β who computes x^{nm}. Now, P_α and P_β together own all the shares i.e. $x^{ij}, i \in [4], j \in [4] \setminus \{i\}$.
Computation: P_α, P_β together run 2PC (instantiated by Yao's protocol [Yao82]) with P_α as GC constructor and P_β as evaluator. P_β computes the output y and sends to all.

Figure 7.2: Protocol $\mathsf{passive2PC}$

Protocol $\mathsf{god4PC}$

Inputs and Output: Party $P_i \in \mathcal{P}$ has x_i. Each party outputs $y = f(x_1, x_2, x_3, x_4)$.
Common Inputs: The circuit C that takes the additive shares x^{ij} of x_i for $i \in [4], j \in [4] \setminus \{i\}$ as inputs and computes $f(x_1, x_2, x_3, x_4)$, each input, their shares and output are from $\{0, 1\}$ (instead of $\{0, 1\}^\ell$ for simplicity).
Notation: $\mathcal{F}$ denotes the two parties identified to be in conflict. $[k]^0, [k]^1$ represent the additive-shares of key k.
Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$, Oblivious Transfer (OT), Garble_3 (Fig 3.9), Eval_3 (Fig 3.10) and collision resistant hash H.
One-time Seed-Distribution: P_1, P_2, P_3 run π_{seedDist} (Fig. 3.7).
Input Commit: $P_i \in \mathcal{P}$ splits its input as $x_i = \oplus_{j \neq i} x^{ij}$, samples pp_i and computes: $(\mathsf{c}_{ij}, \mathsf{o}_{ij}) \leftarrow \mathsf{Com}(\mathsf{pp}_i, x^{ij})$. P_i broadcasts $(\mathsf{pp}_i, \mathsf{c}_{ij})$ and sends o_{ij} privately to P_j. P_j sets $x^{ij} = \mathsf{Open}(\mathsf{pp}_i, \mathsf{c}_{ij}, \mathsf{o}_{ij})$. If o_{ij} is invalid, then P_j sets default value of x^{ij}.
Mask and Blinded Input Transfer:
- For every *input* wire w held by party P_i, each garbler $P_g, g \neq i$ broadcasts $\lambda_w^j, j \in [3] \setminus \mathcal{S}_i$ (if $P_i = P_4$, set $j \in \mathcal{S}_g$). If λ_w^j sent by parties P_α, P_β for $\alpha, \beta \in \mathcal{S}_j$ mismatch, run $\mathsf{passive2PC}$ with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$. Else, P_i uses λ_w^j to compute $\lambda_w = \oplus_{g \in [3]} \lambda_w^g$, sets $b_w = x_w \oplus \lambda_w$ (x_w is the input on w).
- If *input* wire w is owned by two *garblers*, the wire owners (say P_i, P_l) broadcast b_w. If the broadcast values mismatch, then run $\mathsf{passive2PC}$ with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \{P_i, P_l\}$. The blinded bit b_w on wire owned by P_4 is already known to P_4.
- For every *output* wire w, $P_g, g \in [3]$ broadcasts $\lambda_w^j, j \in \mathcal{S}_g$. If λ_w^j sent by parties P_α, P_β for

$\alpha, \beta \in \mathcal{S}_j$ mismatch, run $\mathsf{passive2PC}$ with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$.

Key Transfer: For each *input* wire w, let $\{k^g_{w,0}, k^g_{w,1}\}$ denote two keys derived from seed $\mathsf{s}_g, g \in [3]$.

- For $b \in \{0,1\}$, each $P_g, g \in [3]$ computes commitments for $j \in \mathcal{S}_g$ as: $(c^j_{w,b}, o^j_{w,b}) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k^j_{w,b})$ and broadcasts $(\mathsf{pp}^j, \{c^j_{w,b}\}_{b\in\{0,1\}})$.
- For wire w belonging to share x^{g4} or x^{4g} for $g \in [3]$ and $b \in \{0,1\}$, each $P_j, j \in [3] \setminus \{g\}$ additively shares the key $k^g_{w,b}$ as $k^g_{w,b} = [k^g_{w,b}]^0 \oplus [k^g_{w,b}]^1$. P_j computes $([c^g_{w,b}]^0, [o^g_{w,b}]^0) \leftarrow \mathsf{Com}(\mathsf{pp}^g, [k^g_{w,b}]^0)$ and $([c^g_{w,b}]^1, [o^g_{w,b}]^1) \leftarrow \mathsf{Com}(\mathsf{pp}^j, [k^g_{w,b}]^1)$ and broadcasts $(\mathsf{pp}^g, \{[c^g_{w,b}]^0, [c^g_{w,b}]^1\}_{b\in\{0,1\}})$.
- If $(\mathsf{pp}^g, \{c^g_{w,b}\}_{b\in\{0,1\}})$ or $(\mathsf{pp}^g, \{[c^g_{w,b}]^0, [c^g_{w,b}]^1\}_{b\in\{0,1\}})$ broadcasted by parties P_α, P_β for $\alpha, \beta \in \mathcal{S}_g$ mismatch, run $\mathsf{passive2PC}$ with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$.
- When the input share on w is held by two garblers P_i, P_l where $i < l$, then P_i sends openings $\{o^j_{w,b_w}\}_{j\in\mathcal{S}_i}$ and P_l sends opening $\{o^i_{w,b_w}\}$ (wrt to seed s_i not held by P_i) to P_4. If valid, P_4 uses o^j_{w,b_w} for $j \in [3]$ to compute k^j_{w,b_w}.
- When the input share on w is held by a garbler P_g and P_4 (x^{g4} or x^{4g}), P_g sends openings $\{o^j_{w,b_w}\}_{j\in\mathcal{S}_g}$ to P_4. If valid, P_4 uses o^j_{w,b_w} to compute key k^j_{w,b_w}. The key k^g_{w,b_w} is computed by P_4 as follows: Let $\{\alpha, \beta\} = [3] \setminus \{g\}$.
  - P_g runs a passive OT acting as a receiver with choice bit b_w and P_α acting as sender with inputs $[o^g_{w,0}]^0, [o^g_{w,1}]^0$. Similarly, P_4 runs a passive OT acting as a receiver with choice bit b_w with P_β as sender with inputs $[o^g_{w,0}]^1, [o^g_{w,1}]^1$.
  - P_4 receives $[o^g_{w,b_w}]^1$ as the OT output, and if valid (and indeed corresponds to b_w), computes key-share $[k^g_{w,b_w}]^1$. Similar steps are done by P_g to compute $[k^g_{w,b_w}]^0$ and sends $[o^g_{w,b_w}]^0$ to P_4 which is XORed by P_4 with $[k^g_{w,b_w}]^1$ to obtain k^g_{w,b_w}.

Garbling Phase: Each garbler $P_g, g \in [3]$ runs $\mathsf{Garble}_3(C)$ (Fig 3.9) with π_{3AOTGOD} (Fig. 7.1) to realize OT. P_g broadcasts $\{GC^j\}_{j\in\mathcal{S}_g}$. If any run of π_{3AOTGOD} returns $\mathcal{F}$ or a mismatch occurs in $GC^i, i \in [3]$ sent by P_α, P_β for $\alpha, \beta \in \mathcal{S}_i$, set $\mathcal{F} = \{P_\alpha, P_\beta\}$, then run $\mathsf{passive2PC}$ (Fig 7.2) with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \mathcal{F}$. Else, P_4 sets $GC = GC^1||GC^2||GC^3$.

In all the above cases, if some opening sent by some P_g and received by $P_i, i \in [4]$ (either directly or via OT) is invalid, then P_i broadcasts $(\mathtt{conflict}, P_i, P_g)$ and $\mathsf{passive2PC}$ is run with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \{P_i, P_g\}$. Else, set $\mathbf{X}$ as the set of super-keys for all input wires w i.e. $\{k^g_{w,b_w}\}_{g\in[3]}$.

Evaluation and Output Phase:

- P_4 runs Eval_3 and evaluates GC using $\mathbf{X}$ to obtain output super-key $\mathbf{Y}$ and $z = (y \oplus \lambda_w)$ for output wire w. P_4 unmasks z to compute $y = z \oplus_{g\in[3]} \lambda^g_w$ and outputs y. P_4 broadcasts $\mathbf{Y}$.
- Each garbler P_g accepts $\mathbf{Y}$ if there exists z such that for each $j \in \mathcal{S}_g$, k^j_w obtained from $\mathbf{Y}$ matches $k^j_{w,z}$. P_g outputs $y = z \oplus_{g\in[3]} \lambda^g_w$. Else, $\mathsf{passive2PC}$ is run with parties in $\mathcal{P}^2 = \mathcal{P} \setminus \{P_4, P_3\}$.

Figure 7.3: Protocol $\mathsf{god4PC}$

7.1.1 Optimizations

We use all optimizations of $\mathsf{god5PC}$ to improve the efficiency of our garbling scheme. The use of AOT is optimized by running many AOTs in batches, thus amortizing the communication to 2 commitments and 1 opening per AOT as in [CGMV17]. Further, each DGC fragment is sent by only one garbler privately while the two owners broadcast the hash on it which are compared for equality to determine a conflict, if exists. Likewise, $\mathbf{Y}$ is sent privately to all garblers by P_4 after broadcasting $\mathsf{H}(\mathbf{Y})$. In all cases, broadcast is realized with EIG on the hash of a value rather than the value itself to optimize communication. We use random oracle based instantiations to implement NICOM.

7.2 Properties

Lemma 7.2.1. *The elected 2PC has at most one passive corruption.*

Proof. Let the 2PC be elected after two parties P_α, P_β were identified to be in conflict which could be a consequence of a) P_α, P_β sending conflicting broadcast message or b) one of P_α, P_β raising a conflict against the other for a possibly faulty private communication between the two. In both cases, one of P_α, P_β is actively corrupt party, because if not, then the worst adversarial scenario is one of P_α, P_β is passive, in which case, in a) P_α, P_β would broadcast identical message and in b) no party would send an incorrect private message and the other won't raise a fake conflict. Also, in both the above cases, each message is checked for correctness before proceeding further and thus the conflict could not have been the result of adversary's doing in the previous steps. This implies that the 2PC $\mathcal{P}^2 = \mathcal{P} \setminus \{P_\alpha, P_\beta\}$ does not include the active party. Removing the active party, there remains one passive corruption which can be a part of $\mathcal{P}^2$ in the worst case. $\square$

Lemma 7.2.2. *The output computed by the elected 2PC adheres to the inputs committed in the outer 4PC protocol.*

Proof. A 2PC-instance between $\mathcal{P}^2$ is run after a conflict in the outer 4PC is identified. The inputs in the 2PC are the input-shares as computed in the input commit phase of 4PC. The two parties in $\mathcal{P}^2$ know all input-shares except the two shares that are exclusively owned by the two parties outside $\mathcal{P}^2$. For those two shares, both the parties outside provide share openings (one of which is guaranteed to be correct) to the parties in $\mathcal{P}^2$. Lemma 7.2.1 guarantees honest behavior in the 2PC instance hence ensuring that only the committed inputs are used for computation. $\square$

Lemma 7.2.3. *The protocol* $\mathsf{god4PC}$ *is correct.*

Proof. In case the output is obtained from the 2PC instance, the correctness follows from Lemma 7.2.2 and the correctness of Yao protocol. For an honest execution, when no conflict occurs and the output is obtained from the 4PC itself, the correctness can be argued as: the transfer of input and output wire masks, the masked input, the input keys and the DGC is guaranteed to be correct (as per to the underlying distributed garbling scheme) because of techniques of seed-distribution and *commit publicly, open privately* technique. Otherwise, a conflict would be raised to elect a 2PC, which contradicts our assumption that the output was obtained on decoding $\mathbf{Y}$. Hence, the correctness of $\mathbf{Y}$ and thus the output follows from the correctness of the garbling scheme (Figs 3.9, 3.10). □

Theorem 7.2.4. *The protocol* god4PC *is securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Fig 2.1) in the standard model against an adversary corrupting two parties–* 1 *active,* 1 *passive, assuming enhanced trapdoor permutations.*

Proof. The security proof appears in Section 7.3. □

Although, the formal security proof appears in Section 7.3, here, we provide intuition of GOD for completeness. The input commit phase binds the adversary to commit to an input or a default value. If a conflict is identified at any point during the execution, then an elected 2PC committee runs passive 2PC [Yao82] to obtain the output y. Otherwise, computation proceeds as per the honest run and each party receives the output using the $\mathbf{Y}$ broadcasted by P_4. If $\mathbf{Y}$ is valid, then all parties compute y using $\mathbf{Y}$ to conclude the execution. Else if $\mathbf{Y}$ is invalid or not received, a 2PC instance is identified among the garblers to compute y. In both the above cases (Lemma 7.2.2), the inputs committed in input phase alone are used to obtain the output y thus concluding the intuition.

7.3 Security Proof of god4PC

In this section, we outline the complete security proof of Theorem 7.2.4 that describes the security of our god4PC protocol relative to its ideal functionality in the standard security model in the $\mathcal{F}_{\mathsf{OT}}$ hybrid model.

Proof. We describe the simulator $\mathcal{S}_{\mathsf{god4PC}}$ for three cases which exhaustively cover the corruption scenarios: First, when P_1 is actively corrupt and P_2 is passively corrupt. Second, when P_1 is actively corrupt and P_4 is passively corrupt. Finally, when P_4 is actively corrupt and P_1 is passively corrupt. The corruption of any two garblers is symmetric to the case when P_1, P_2 are corrupt, the corruption of any one actively corrupt garbler and passively corrupt evaluator is symmetric to the second case and the corruption of any one passively corrupt garbler and actively corrupt evaluator is symmetric to the third case. The simulator acts on behalf of all honest parties in the execution. For better understanding we separate out the simulation for the subroutine π_{seedDist} from the simulation of main protocol.

We give a high level view of the simulation of garbling and output computation as follows: First, in the case of P_1^* actively corrupt and P_2° passively corrupt, the evaluator P_4 is honest. Hence, in this case, correctness is required from the distributed GC. The simulator behaves as an honest P_3 by raising conflicts as per the protocol in case of any cheating throughout the garbling phase, since all seeds are known to the adversary. If no cheating is detected throughout the GC construction, then a GC is generated as per the Garble_3 procedure. Else a 2PC instance is identified and the 4PC simulator in turn invokes the simulator of 2PC [Yao82] protocol to complete the simulation. Second, in the case of actively corrupt P_1^* and passively corrupt P_4°, the simulator knows the seeds held by the adversary. In addition the simulator has complete control over the part of GC generated using the seed s_1. Since input extraction of actively corrupt P_1^* is done in the input commit phase and in the execution of OTs, the simulator can invoke the functionality to obtain y in advance at the time of garbling. As a result with the knowledge of y, a fake garbled circuit is constructed by the simulator using s_1 that always evaluates to the same output keys forming the output super-key $\mathbf{Y}$, which correspond to the evaluation performed using the extracted inputs of the adversary and the inputs of the honest parties. Finally, $\mathbf{Y}$ is received from P_4° on behalf of honest parties, then the simulation terminates. (Since the simulator for 2PC is already well-described in [LP04], we do not provide details of it). A similar strategy as explained in the second case is employed for the case when P_1° is passively corrupt and P_4^* is actively corrupt except that the input of P_1° is available at the onset and the input of P_4^* is extracted from the input commit phase and OTs. Finally, if $\mathbf{Y}$ is received from P_4^* on behalf of honest parties then the simulation terminates, else a 2PC instance is identified according to the protocol and the 4PC simulator runs the simulator of the 2PC instance sub-routine to complete the simulation.

We describe the simulator steps in detail for π_{seedDist} and the main protocol separately in Figs 7.4, 7.6, 7.8 and Figs7.5, 7.7, 7.5 respectively.

Simulator $\mathcal{S}^{1\mathsf{A},2\mathsf{P}}_{\pi_{\mathsf{seedDist}}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1^* as sender and P_3 as receiver to obtain seed s_2.

- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2° as receiver.

- For the commitment instance between P_1^* as sender and P_2° as receiver to commit to seed s_3:
    - Run the ExtCom protocol where P_1^* and P_2° run rounds 1-3 and broadcast their messages $(\mathsf{extcom}_1^1, \mathsf{extcom}_2^1, \mathsf{extcom}_3^1)$.
    - Rewind the adversary to the end of round 1 for P_1^* and P_2° to rerun rounds 2-3 and broadcast $(\mathsf{extcom}_2^2, \mathsf{extcom}_3^2)$.
    - On behalf of P_3, Run extractor algorithm $\mathsf{Extract}$ of the commitment scheme as in Fig 2.4 using inputs $(\mathsf{extcom}_1^1, \{\mathsf{extcom}_2^i, \mathsf{extcom}_3^i\}_{i\in[2]})$ to extract the committed seed s_3.

Figure 7.4: Simulator $\mathcal{S}_{\pi_{\mathsf{seedDist}}}^{\mathsf{1A,2P}}$ for π_{seedDist} with actively corrupt P_1^* and passively corrupt P_2°

Simulator $\mathcal{S}_{\mathsf{god4PC}}^{\mathsf{1A,2P}}$

Seed Distribution Phase (one-time): Invoke $\mathcal{S}_{\pi_{\mathsf{seedDist}}}^{\mathsf{1A,2P}}$ (Fig 7.4). Extract s_3.

Input Distribution Phase: Obtain x_2 as the input provided to simulator.

For input of active P_1^* (x_1):

- Receive $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ as broadcasted by P_1^* on behalf of the honest parties. Receive o_{1i} on behalf of $P_i, i \in \{3,4\}$ and compute $x^{1i} \leftarrow \mathsf{Open}(\mathsf{pp}_1, c_{1i}, o_{1i})$. If o_{1i} is invalid, set x^{1i} to the default value.

For input of passive P_2° (x_2):

- Receive $(\mathsf{pp}_2, c_{21}, c_{23}, c_{24})$ as broadcasted by P_2° on behalf of the honest parties. Receive o_{2i} on behalf of $P_i, i \in \{3,4\}$ and compute $x^{2i} \leftarrow \mathsf{Open}(\mathsf{pp}_2, c_{2i}, o_{2i})$.

For input of honest P_3 (x_3):

- On behalf of P_3: sample random x^{31}, x^{32} and compute commitments as $(c_{3i}, o_{3i}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, x^{3i})$ for $i \in [2]$. Choose a dummy commitment c_{34}. Broadcast $(\mathsf{pp}_3, c_{31}, c_{32}, c_{34})$ and send o_{31}, o_{32} privately to P_1^*, P_2° respectively. Similar steps are done for honest P_4's input.

Mask and Blinded Input Transfer:

- On behalf of P_3 do the following: For every *input* wire w with party P_i holding the value on wire w, broadcast $\lambda_w^j, j \in \mathcal{S}_3 \setminus \mathcal{S}_i$ (for P_4, set $j \in \mathcal{S}_3$). Send o_w^j privately to P_i. If λ_w^j sent by parties $P_1^*, P_l \in \mathcal{S}_1$ mismatch, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1, P_l\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For input wire w owned by P_1^* and P_2° (say, corresponding to share x^{12}): receive b_w ($b_w = x_w \oplus \lambda_w$ where x_w is the bit on wire w) as broadcasted by P_1^* and P_2°. If mismatching values are broadcasted, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_2^\circ\}$ to complete

the simulation and send the output y to all on behalf of honest parties. Else, compute $x^{12} = b_w \oplus (\oplus_{i\in[3]}\lambda_w^i)$ (using the knowledge of all seeds). Compute $x_1 = x^{12} \oplus x^{13} \oplus x^{14}$. Invoke $\mathcal{F}_{\mathsf{god}}$ (FIg 2.1) with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_2)$ on behalf of corrupt P_1^*, P_2° to obtain y. Similar steps are done for the input share x^{21} held by P_1^*, P_2°.

- For input wire w owned by P_1^* and P_3 (say, corresponding to share x^{13}): Broadcast b_w ($b_w = x_w \oplus \lambda_w$ where x_w is the bit on wire w) on behalf of P_3. Also receive b_w as broadcasted by P_1^* on behalf of the honest parties. If mismatching values are broadcasted, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_3\}$ to complete the simulation and send the output y to all on behalf of honest parties. Similar steps are done for the input share x^{21} held by P_1^*, P_3.
- For input wire w owned by P_2° and P_3 (say, corresponding to share x^{23}): Broadcast b_w ($b_w = x_w \oplus \lambda_w$ where x_w is the bit on wire w) on behalf of P_3. Also receive b_w as broadcasted by P_2° on behalf of the honest parties. Similar steps are done for the input share x^{32} held by P_2°, P_3.
- For every *output* wire w, broadcast $\lambda_w^h, h \in \mathcal{S}_3$ on behalf of P_3. If $h \in \mathcal{S}_1$ and P_1^* broadcasts a mismatching λ_w^h (in comparison to λ_w^h broadcast by honest/passive P_i), invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_3\}$ to complete the simulation and send the output y to all on behalf of honest parties.

Key Transfer: For every *input* wire w, let $\{k_{w,0}^g, k_{w,1}^g\}$ denote the two keys derived from seed s_g for $g \in [3]$.

- On behalf of P_3: for $b \in \{0,1\}, j \in \mathcal{S}_3$, compute commitments as: $(c_{w,b}^j, o_{w,b}^j) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k_{w,b}^j)$ and broadcast $(\mathsf{pp}^j, \{c_{w,b}^j\}_{b\in\{0,1\}})$.
- On behalf of P_3: for input wire w corresponding to share x^{g4} or x^{4g} for $g \in [2]$ and $b \in \{0,1\}$, split key $k_{w,b}^g = [k_{w,b}^g]^0 \oplus [k_{w,b}^g]^1$. Compute $([c_{w,b}^g]^0, [o_{w,b}^g]^0) \leftarrow \mathsf{Com}(\mathsf{pp}^g, [k_{w,b}^g]^0)$ and $([c_{w,b}^g]^1, [o_{w,b}^g]^1) \leftarrow \mathsf{Com}(\mathsf{pp}^j, [k_{w,b}^g]^1)$ and broadcasts $(\mathsf{pp}^g, \{[c_{w,b}^g]^0, [c_{w,b}^g]^1\}_{b\in\{0,1\}})$.
- If $(\mathsf{pp}^g, \{c_{w,b}^g\}_{b\in\{0,1\}})$ or $(\mathsf{pp}^g, \{[c_{w,b}^g]^0, [c_{w,b}^g]^1\}_{b\in\{0,1\}})$ broadcasted by parties in $\mathcal{S}_1$ mismatch, add parties in $\mathcal{S}_1$ to $\mathcal{F}$ and invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \mathcal{F}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For the input wire w owned by P_1^* and P_2°, receive openings $\{o_{w,b_w}^j\}_{j\in\mathcal{S}_1}$ from P_1^* and $\{o_{w,b_w}^1\}$ from P_2 on behalf of P_4. If opening sent by P_1 is invalid, broadcast $(\mathtt{conflict}, P_1^*, P_4)$. Invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_4\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For the input wire w owned by P_3 and garbler P_1^*, receive openings $\{o_{w,b_w}^j\}_{j\in\mathcal{S}_g}$ sent by P_1^* on behalf of P_4. If opening sent by P_1^* is invalid, broadcast $(\mathtt{conflict}, P_1^*, P_4)$. Invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_4\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For the input wire w owned by P_3 and semi-honest P_2°, receive openings $\{o_{w,b_w}^j\}_{j\in\mathcal{S}_g}$ sent by P_2°

on behalf of P_4.

- For input wire w held by the adversary $P_g, g \in [2]$ and P_4, receive openings $\{o^j_{w,b_w}\}_{j \in \mathcal{S}_g}$ from P_g on behalf of P_4 while for opening $\{o^g_{w,b_w}\}$:
  - ∘ Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_g (as receiver) and $P_h, h \in [2] \backslash \{g\}$ (as sender). If P_g broadcasts ($\texttt{conflict}, P^*_g$, P_h), invoke simulator of $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P^*_g, P_h\}$ to complete the simulation and send the output y to all on behalf of honest parties.
  - ∘ Receive $[o^g_{w,b_w}]^0$ from P_g on behalf of P_4. For $g = 1$, if the opening is invalid, broadcast ($\texttt{conflict}, P^*_1, P_4$), invoke simulator of $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P^*_1, P_4\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For input wire w held by a garbler P_3 and evaluator P_4, do the following for opening $\{o^3_{w,b_w}\}$:
  - ∘ Invoke $\mathcal{F}_{\mathsf{OT}}$ on behalf of P_3 (as receiver) and P^*_1 (as sender) to obtain $[o^3_{w,b_w}]^0$. If invalid, broadcast ($\texttt{conflict}, P_3, P^*_1$) on behalf of P_3 and invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P^*_1, P_3\}$ to complete the simulation and send the output y to all on behalf of honest parties.
  - ∘ Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ on behalf of P_4 (as receiver) and P°_2 (as sender) to obtain $[o^3_{w,b_w}]^1$.

Garbling Phase:

- Behave honestly on behalf of P_3 in Garble_3 and Π_{3AOTGOD} using seeds chosen in seed distribution phase. If any run of Π_{3AOTGOD} returns $\mathcal{F}$ (because of misbehaviour by P^*_1), invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \mathcal{F}$ to complete the simulation and send the output y to all on behalf of honest parties.
- Broadcast GC^h for $h \in \mathcal{S}_3$ on behalf of P_3. Receive GC^g as broadcasted by $P_i, i \in [2]$ for $g \in \mathcal{S}_i$. If a mismatch occurs in $GC^i, i \in \mathcal{S}_1$ sent by parties in $\mathcal{S}_1$, add parties in $\mathcal{S}_1$ to $\mathcal{F}$ and invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \mathcal{F}$ to complete the simulation and send the output y to all on behalf of honest parties. Else, on behalf of P_4, set $GC = GC^1 || GC^2 || GC^3$.

Evaluation and Output Phase:

- Using the knowledge of all seeds $\mathsf{s}_g, g \in [3]$ and y set $z = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,z}\}_{g \in [3]}$ for output wire w. Broadcast $\mathbf{Y}$ on behalf of P_4 to complete the simulation.

Figure 7.5: Simulator $\mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{god4PC}}$ for $\mathsf{god4PC}$ with actively corrupt P^*_1 and passively corrupt P°_2.

The hybrid arguments are as follows:

Security against actively corrupt P^*_1 and passively corrupt P°_2: We now formally argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{god4PC}}} \stackrel{c}{\approx} \textsc{real}_{\mathsf{god4PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P^*_1 actively and P°_2 passively. The views are shown to be indistinguishable via a series of intermediate hybrids.

– $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{god4PC}, \mathcal{A}}$.

- HYB_1: Same as HYB_0 except: rerun rounds 2-3 of extractable commitment (with P_2° as sender and P_1^* as receiver) in the seed-distribution phase to extract seed s_3. Run the subsequent rounds same as HYB_0.

- HYB_2: Same as HYB_1 except that for share x^{34} (i.e. the share that the adversary doesn't get access to), replace c_{34} with the commitment of a dummy value in input commit phase. Do the same for share x^{43}.

- HYB_3: Same as HYB_2 except that P_4 raises a conflict to identify a 2PC instance if any decommitment for $\{k^g_{w,0}, k^g_{w,1}\}_{g\in[3]}$ corresponding to a committed share opens to a value other than what was originally committed and held by P_1^*.

- HYB_4: Same as HYB_3 except: for input wire w held by garbler (say P_3) and P_4, to obtain opening o^3_{w,b_w}, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_3 as receiver and P_1^* as sender to obtain $[o^3_{w,b_w}]^0$. Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4 as receiver and P_2° as sender to obtain $[o^3_{w,b_w}]^1$.

- HYB_5: Same as HYB_4 except: for wire w with share x^{12} owned by P_1^* and P_2° use b_w to obtain $x^{12} = b_w \oplus_{i\in[3]} \lambda^i_w$ (using knowledge of all seeds) and compute $x_1 = x^{12} \oplus x^{13} \oplus x^{14}$. Invoke the ideal functionality $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_2)$ to obtain y. Compute $z = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,z}\}_{g\in[3]}$ instead of running the Evaluation Phase of garbling.

- HYB_6: Same as HYB_5 except: in case of a 2PC instance elected because of a public/private conflict, invoke simulator for $\mathsf{passive2PC}$ of [Yao82] presented in [LP04] instead of running $\mathsf{passive2PC}$.

Note that $\text{HYB}_6 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{god4PC}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference between the hybrids is that, in HYB_1, rounds 2-3 of the extractable commitment are rewound. However, the adversary's view contains only the final rewound execution and the previous rewinds are erased. Hence the hybrids HYB_0 and HYB_1 are indistinguishable.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that in HYB_2, the commitment for shares x^{34} and x^{43} are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_3$: The only difference between the hybrids is that in $\textsc{hyb}_3$, P_4 raises a conflict if the decommitment for $\{k^g_{w,0}, k^g_{w,1}\}_{g\in[3]}$ corresponding to a committed share and sent by P_1^* is invalid (the decommitment is $\perp$) whereas in $\textsc{hyb}_2$, P_4 raises a conflict to identify the 2PC instance if the decommitment corresponding a committed share opens to a value other than what was originally committed and held by P_1^*. Since the commitment scheme Com is binding for any pp, P_1^* could have successfully decommitted to a value than what was originally committed with negligible probability. Hence, the hybrids are indistinguishable.

$\textsc{hyb}_3 \overset{c}{\approx} \textsc{hyb}_4$: Indistinguishability of hybrids follows from the security of the underlying OT scheme [EGL85].

$\textsc{hyb}_4 \overset{c}{\approx} \textsc{hyb}_5$: The only difference between the hybrids is that, in $\textsc{hyb}_4$, $\mathbf{Y}$ is computed as $\mathbf{Y} = \{k^g_{w,y_w\oplus\lambda_w}\}_{g\in[3]}$ instead of running the Evaluation Phase of the garbling. The indistinguishability follows from the correctness of the garbling scheme (follows from Lemma 3.1.4) since $\mathbf{Y}$ computed using the Evaluation Phase of garbling would also result in $\mathbf{Y} = \{k^g_{w,y\oplus\lambda_w}\}_{g\in[3]}$ where $y = f(x_1, x_2, x_3, x_4)$ except with negligible probabiltiy.

$\textsc{hyb}_5 \overset{c}{\approx} \textsc{hyb}_6$: The indistinguishability follows from the indistinguishability of $\mathsf{passive2PC}$ simulator $\mathcal{S}_{\mathsf{passive2PC}}$ (by the security of [Yao82] presented in [LP04]) with the real execution of [Yao82].

We now describe the simulator and hybrid arguments for the second case.

Protocol $\mathcal{S}^{\mathsf{1A,4P}}_{\pi_{\mathsf{seedDist}}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1^* as sender and P_3 as receiver to obtain seed s_2. Abort if P_1 sends incorrect opening.
- Sample random s_3 and act honestly on behalf of P_2 for the commitment instance between P_2 as sender and P_1^* as receiver.
- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2 as receiver.

Figure 7.6: Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\pi_{\mathsf{seedDist}}}$ for π_{seedDist} with actively corrupt P_1^* and passively corrupt P_4°

Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\mathsf{god4PC}}$

Seed Distribution Phase (one-time): Invoke $\mathcal{S}^{\mathsf{1A,4P}}_{\pi_{\mathsf{seedDist}}}$ (Fig 7.6).

Input Distribution Phase: Obtain x_4 as the input provided to simulator.

For input of active P_1^* (x_1):

- Receive $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ as broadcasted by P_1^*. Receive o_{1i} on behalf of $P_i, i \in \{2, 3\}$ and compute $x^{1i} \leftarrow \mathsf{Open}(\mathsf{pp}_1, c_{1i}, o_{1i})$. If o_{1i} is invalid, set x^{1i} to the default value.

For input of passive P_4° (x_4):

- Receive $(\mathsf{pp}_4, c_{41}, c_{42}, c_{43})$ as broadcasted by P_4°. Receive o_{4i} on behalf of $P_i, i \in \{2, 3\}$ and compute $x^{4i} \leftarrow \mathsf{Open}(\mathsf{pp}_4, c_{4i}, o_{4i})$.

For input of honest P_3 (x_3):

- On behalf of P_3: sample random x^{31}, x^{34} and compute commitment as $(c_{3i}, o_{3i}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, x^{3i})$ for $i \in \{1, 4\}$. Choose a dummy commitment c_{32} for x^{32}. Broadcast $(\mathsf{pp}_3, c_{31}, c_{32}, c_{34})$ and send o_{3i} privately to P_i. Similar steps are done for the input of P_2.

Mask and Blinded Input Transfer:

- On behalf of $P_g, g \in \{2, 3\}$ do the following: For every *input* wire w with party P_i holding the value on wire w, broadcast $\lambda_w^j, j \in \mathcal{S}_g \setminus \mathcal{S}_i$ (for P_4, set $j \in \mathcal{S}_g$). If λ_w^j sent by parties P_1^*, P_l in $\mathcal{S}_1$ mismatch, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_l\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For input wire w owned by P_1^* and $P_g, g \in \{2, 3\}$ do the following on behalf of P_g: Compute $\lambda_w = \oplus_{h \in [3]} \lambda_w^h$ and $b_w = x_w \oplus \lambda_w$ where x_w is the bit on wire w and broadcast b_w on behalf of P_g. Also receive b_w as broadcasted by P_1^* on behalf of the honest parties. If mismatching values are broadcasted, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_g\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For input wire w owned by P_2 and P_3 do the following on behalf of $P_g, g \in \{2, 3\}$: Compute $\lambda_w = \oplus_{h \in [3]} \lambda_w^h$ and $b_w = x_w \oplus \lambda_w$ where x_w is a dummy value $(= 0)$ for share on wire w. Broadcast b_w on behalf of honest parties.
- For every *output* wire w, broadcast $\lambda_w^j, j \in \mathcal{S}_1$ on behalf of honest$P_h, h \in \mathcal{S}_1$ respectively. If P_1^* broadcasts a mismatching λ_w^j, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_h\}$ to complete the simulation and send the output y to all on behalf of honest parties.

Key Transfer: For every *input* wire w, let $\{k_{w,0}^g, k_{w,1}^g\}$ denote the two keys derived from seed s_g for $g \in [3]$.

- On behalf of $P_g, g \in \{2, 3\}$: for $b \in \{0, 1\}, j \in \mathcal{S}_g$, compute commitments as: $(c_{w,b}^j, o_{w,b}^j) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k_{w,b}^j)$ and broadcast $(\mathsf{pp}^j, \{c_{w,b}^j\}_{b \in \{0,1\}})$.
- On behalf of $P_h, h \in \{2, 3\}$: for input wire w corresponding to share held by P_4° and a garbler P_g and $b \in \{0, 1\}$, split key $k_{w,b}^g$ as $k_{w,b}^g = [k_{w,b}^g]^0 \oplus [k_{w,b}^g]^1$. Compute $([c_{w,b}^g]^0, [o_{w,b}^g]^0) \leftarrow$

$\mathsf{Com}(\mathsf{pp}^g, [k^g_{w,b}]^0)$, $([c^g_{w,b}]^1, [o^g_{w,b}]^1) \leftarrow \mathsf{Com}(\mathsf{pp}^j, [k^g_{w,b}]^1)$ and broadcasts $(\mathsf{pp}^g, \{[c^g_{w,b}]^0, [c^g_{w,b}]^1\}_{b\in\{0,1\}})$.

- If P_1^* broadcasts $(\mathsf{pp}^g, \{c^g_{w,b}\}_{b\in\{0,1\}})$ or $(\mathsf{pp}^g, \{[c^g_{w,b}]^0, [c^g_{w,b}]^1\}_{b\in\{0,1\}})$ is different from that broadcasted by P_h for $\{1,h\} \in \mathcal{S}_g$, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_h\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For the input wire w owned by P_1^* and $P_g, g \in \{2,3\}$, send opening $\{o^1_{w,b_w}\}$ to P_4° on behalf of P_g.
- For the input wire w owned by P_2 and P_3, send openings $\{o^j_{w,b_w}\}_{j\in\mathcal{S}_2}$ on behalf of P_2 and opening o^2_{w,b_w} on behalf of P_3 to P_4°.
- For input wire w held by a garbler P_1^* and P_4° (say x^{14}), the following is done for opening $\{o^1_{w,b_w}\}$:
  - ∘ Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_1^* (as receiver) and P_2 (as sender). If P_1^* broadcasts $(\mathtt{conflict}, P_1^*, P_2)$, invoke simulator of $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P}\setminus\{P_1^*, P_2\}$ to complete the simulation and send the output y to all on behalf of honest parties.
  - ∘ Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4° (as receiver) and P_3 (as sender). Obtain receiver's choice bit b_w sent by P_4° to $\mathcal{F}_{\mathsf{OT}}$. Compute $x^{14} = b_w \oplus (\oplus_{g\in[3]}\lambda^g_w)$. Compute $x_1 = x^{12} \oplus x^{13} \oplus x^{14}$. Invoke $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_4)$ on behalf of corrupt P_1^*, P_4° to obtain y.

  Similar steps are done for input share x^{41}.
- For input wire w held by a garbler $P_g, g \in \{2,3\}$ and P_4°, do the following for opening $\{o^g_{w,b_w}\}$:
  - ∘ Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_g (as receiver) and P_1^* (as sender) to obtain $[o^g_{w,b_w}]^0$. If invalid, broadcast $(\mathtt{conflict}, P_g, P_1^*)$ on behalf of P_g and invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_1^*, P_g\}$ to complete the simulation and send the output y to all on behalf of honest parties. Else, send $[o^g_{w,b_w}]^0$ on behalf of P_g to P_4°.
  - ∘ Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4° (as receiver) and $P_h, h \in [3] \setminus \{1, g\}$ (as sender).

Garbling Phase:

- Using knowledge of s_1 (which is not known to the adversary) and output y, behave in Garble_3 and Π_{3AOTGOD} in such a way that each ciphertext for the output gate of GC^g for $g \in [3]$ encrypts the same output key $k^g_{w,z}$ where $z = y \oplus \lambda_w$.
- If any run of Π_{3AOTGOD} returns $\mathcal{F}$ (because of misbehaviour by P_1^*), invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ to complete the simulation and send the output y to all on behalf of honest parties.
- Broadcast GC^h for $h \in \mathcal{S}_g$ on behalf of $P_g, g \in \{2,3\}$. Receive GC^g as broadcasted by P_1^* for $g \in \mathcal{S}_1$ on behalf of honest parties. If a mismatch occurs in GC^g sent by parties in $\mathcal{S}_1$, add parties in $\mathcal{S}_1$ to $\mathcal{F}$ and invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \mathcal{F}$ to complete the simulation and send the output y to all on behalf of honest parties.

Evaluation and Output Phase:

- Receive $\mathbf{Y}$ on behalf of $P_g, g \in \{2,3\}$ as broadcasted by P_4°. Output y.

Figure 7.7: Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\mathsf{god4PC}}$ for $\mathsf{god4PC}$ with actively corrupt P_1^* and passively corrupt P_4°.

Security against actively corrupt P_1^* and passively corrupt P_4°: We now formally argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^{\mathsf{1A,4P}}_{\mathsf{god4PC}}} \stackrel{c}{\approx} \textsc{real}_{\mathsf{god4PC},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1 actively and P_4 passively. The views are shown to be indistinguishable via a series of intermediate hybrids.

- $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{god4PC},\mathcal{A}}$.
- $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except: for share x^{23} (i.e. the share that the adversary doesn't get access to), replace c_{23} with the commitment of a dummy value in input commit phase. Do the same for share x^{32}.
- $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except: for wire w with share x^{23} and x^{32} (i.e. shares held only by the honest parties) assume a dummy value ($= 0$), compute commitment (c_w, o_w) on $b_w = x_w \oplus \lambda_w$ using seed s_1.
- $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that some of the commitments of input keys sent by P_2, P_3 wrt seed s_1, which will not be opened are replaced with commitments of dummy values. These commitments correspond to the labels that do not correspond to any input share.
- $\textsc{hyb}_4$: Same as $\textsc{hyb}_3$ except: invoke $\mathcal{F}_{\mathsf{OT}}$ appropriately for the transfer of openings of key-shares corresponding to input wire w owned by garbler $P_g, g \in [3]$ and evaluator P_4°.
- $\textsc{hyb}_5$: Same as $\textsc{hyb}_4$ except: instead of constructing an honest GC, a simulated GC is constructed using the knowledge of seed s_1 (not known to the adversary), in such a way that each ciphertext for the output gate encrypts the same output key which corresponds to $z = y \oplus \lambda_w$ where y is obtained after having invoked $\mathcal{F}_{\mathsf{god}}$ and λ_w is known from the information of all seeds.
- $\textsc{hyb}_6$: Same as $\textsc{hyb}_5$ except: in case of a 2PC instance elected because of a public/private conflict, invoke simulator for $\mathsf{passive2PC}$ as in [LP04] instead of running $\mathsf{passive2PC}$.

Note that $\textsc{hyb}_6 = \textsc{ideal}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{god4PC}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:
$\textsc{hyb}_0 \stackrel{c}{\approx} \textsc{hyb}_1$: The only difference between the hybrids is that in $\textsc{hyb}_1$, the commitment for shares x^{23} and x^{32} are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_1 \overset{c}{\approx} \textsc{hyb}_2$: The only difference in the value of b_w computed such that in $\textsc{hyb}_1$, it is w.r.t. honest share x_w while in $\textsc{hyb}_2$, it is w.r.t. dummy share 0. This remains indistinguishable to the adversary as she is unaware of seed s_1 and hence can't compute the underlying x_w.

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_3$: The only difference between the hybrids is that, in $\textsc{hyb}_3$, the commitments of input wire labels wrt seed s_1, which will not be opened are replaced with commitments on dummy values. The indistingushability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_3 \overset{c}{\approx} \textsc{hyb}_4$: Indistinguishability of hybrids follows from reduction to the security of the underlying OT scheme [EGL85].

$\textsc{hyb}_4 \overset{c}{\approx} \textsc{hyb}_5$: Indistinguishability follows from reduction to the security of the underlying garbling scheme which breaks down to the security of PRF.

$\textsc{hyb}_5 \overset{c}{\approx} \textsc{hyb}_6$: The indistinguishability follows from the indistinguishability of passive2PC simulator (follows from the security of [Yao82] provided in [LP04]) to the real execution of [Yao82].

We now describe the simulator and hybrid arguments for the final case.

Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\pi_{\mathsf{seedDist}}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1° as sender and P_3 as receiver to obtain seed s_2.
- Sample random s_3 and act honestly on behalf of P_2 for the commitment instance between P_2 as sender and P_1° as receiver.
- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2 as receiver.

Figure 7.8: Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\pi_{\mathsf{seedDist}}}$ for π_{seedDist} with actively corrupt P_4^* and passively corrupt P_1°

Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\mathsf{god4PC}}$

Seed Distribution Phase (one-time): Invoke $\mathcal{S}^{\mathsf{4A,1P}}_{\pi_{\mathsf{seedDist}}}$ (Fig 7.8).

Input Distribution Phase: Obtain x_1 as the input provided to simulator.

For input of active P_4^* (x_4):

- Receive $(\mathsf{pp}_4, c_{41}, c_{42}, c_{43})$ as broadcasted by P_4^*. Receive o_{4i} on behalf of $P_i, i \in \{2,3\}$ and compute $x^{4i} \leftarrow \mathsf{Open}(\mathsf{pp}_4, c_{4i}, o_{4i})$.

For input of passive P_1° (x_1):

- Receive $(\mathsf{pp}_1, c_{12}, c_{13}, c_{14})$ as broadcasted by P_1^*. Receive o_{1i} on behalf of $P_i, i \in \{2,3\}$ and compute $x^{1i} \leftarrow \mathsf{Open}(\mathsf{pp}_1, c_{1i}, o_{1i})$.

For input of honest P_3 (x_3):

- On behalf of P_3: sample random x^{31}, x^{34} and compute commitment as $(c_{3i}, o_{3i}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, x^{3i})$ for $i \in \{1,4\}$. Broadcast $(\mathsf{pp}_3, c_{31}, c_{32}, c_{34})$ and send o_{3i} privately to P_i. Similar steps are done for input x_2.

Mask and Blinded Input Transfer:

- On behalf of $P_g, g \in \{2,3\}$ do the following: For every *input* wire w with party P_i holding the value on wire w, broadcast $\lambda_w^j, j \in \mathcal{S}_g \setminus \mathcal{S}_i$ (for P_4^*, set $j \in \mathcal{S}_g$).
- For input wire w owned by P_1° and $P_g, g \in \{2,3\}$ do the following on behalf of P_g: Compute $\lambda_w = \oplus_{h \in [3]} \lambda_w^h$ and $b_w = x_w \oplus \lambda_w$ where x_w is the bit on wire w and broadcast b_w on behalf of P_g. Also receive b_w as broadcasted by P_1° on behalf of P_g.
- For input wire w owned by P_2 and P_3 do the following on behalf of $P_g, g \in \{2,3\}$: Compute $\lambda_w = \oplus_{h \in [3]} \lambda_w^h$ and $b_w = x_w \oplus \lambda_w$ where x_w is a dummy value ($= 0$) for share on wire w. Broadcast b_w on behalf of honest parties.
- For every *output* wire w, broadcast $\lambda_w^h, h \in \mathcal{S}_g$ on behalf of $P_g, g \in \{2,3\}$.

Input and Key Transfer: For every *input* wire w, let $\{k_{w,0}^g, k_{w,1}^g\}$ denote the two keys derived from seed $\{\mathsf{s}_g\}$ for $g \in [3]$.

- On behalf of $P_g, g \in \{2,3\}$: for $b \in \{0,1\}, j \in \mathcal{S}_g$, compute commitments as: $(c_{w,b}^j, o_{w,b}^j) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k_{w,b}^j)$ and broadcast $(\mathsf{pp}^j, \{c_{w,b}^j\}_{b \in \{0,1\}})$.
- On behalf of $P_h, h \in \{2,3\}$: for input wire w corresponding to share held by P_4^* and a garbler P_g and $b \in \{0,1\}$, split key $k_{w,b}^g$ as $k_{w,b}^g = [k_{w,b}^g]^0 \oplus [k_{w,b}^g]^1$. Compute $([c_{w,b}^g]^0, [o_{w,b}^g]^0) \leftarrow \mathsf{Com}(\mathsf{pp}^g, [k_{w,b}^g]^0)$ and $([c_{w,b}^g]^1, [o_{w,b}^g]^1) \leftarrow \mathsf{Com}(\mathsf{pp}^j, [k_{w,b}^g]^1)$, broadcasts $(\mathsf{pp}^g, \{[c_{w,b}^g]^0, [c_{w,b}^g]^1\}_{b \in \{0,1\}})$. Also, receive the same from P_1°.
- For the input wire w owned by P_1° and $P_g, g \in \{2,3\}$, send opening $\{o_{w,b_w}^1\}$ to P_4^* on behalf of P_g. If P_4^* broadcasts $(\texttt{conflict}, P_4^*, P_g/P_1^\circ)$, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_4^*, P_g/P_1^\circ\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For the input wire w owned by P_2 and P_3, send openings $\{o_{w,b_w}^j\}_{j \in \mathcal{S}_2}$ on behalf of P_2 and opening o_{w,b_w}^2 on behalf of P_3 to P_4^*. If P_4^* broadcasts $(\texttt{conflict}, P_4^*, P_g)$ for $g \in \{2,3\}$, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_4^*, P_g\}$ to complete the simulation and send the output

y to all on behalf of honest parties.

- For input wire w held by a garbler P_1° and P_4^* (say x^{41}), the following is done for opening $\{o^1_{w,b_w}\}$:
  - ◦ Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_1° (as receiver) and P_2 (as sender). Obtain receiver's choice bit b_w sent by P_1° to $\mathcal{F}_{\mathsf{OT}}$. Compute $x^{41} = b_w \oplus (\oplus_{g \in [3]} \lambda_w^g)$. Compute $x_4 = x^{41} \oplus x^{42} \oplus x^{43}$. Invoke $\mathcal{F}_{\mathsf{god}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_4)$ on behalf of corrupt P_1°, P_4^* to obtain y.
  - ◦ Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4^* (as receiver) and P_3 (as sender). If P_4^* broadcasts ($\mathtt{conflict}, P_4^*$, P_3) , invoke simulator for *TwoPC*, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_4^*, P_3\}$ to complete the simulation and send the output y to all on behalf of honest parties.
- For input wire w held by a garbler $P_g, g \in \{2, 3\}$ and P_4^*, do the following for opening $\{o^g_{w,b_w}\}$:
  - ◦ Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_g (as receiver) and P_1° (as sender) to obtain $[o^g_{w,b_w}]^0$. Send $[o^g_{w,b_w}]^0$ on behalf of P_g to P_4^*.
  - ◦ Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4^* (as receiver) and $P_h, h \in [3] \setminus \{1, g\}$ (as sender). If P_4^* broadcasts $(\mathtt{conflict}, P_4^*, P_h)$, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_4^*, P_h\}$ to complete the simulation and send the output y to all on behalf of honest parties.

Garbling Phase:

- Compute $z = y \oplus \lambda_w$ for the output wire w. Using knowledge of s_1 (which is not known to the adversary) and output y, behave in Garble_3 and Π_{3AOTGOD} in such a way that each ciphertext for the output gate of GC^g for $g \in [3]$ encrypts the same output key $k^g_{w,z}$.
- On behalf of $P_g, g \in \{2, 3\}$: Broadcast GC^h for $h \in \mathcal{S}_g$. Also, receive $GC^j, j \in \mathcal{S}_1$ broadcast by P_1° on behalf of the honest parties.

Evaluation and Output Phase:

- Receive $\mathbf{Y}$ on behalf of $P_g, g \in \{2, 3\}$ as broadcasted by P_4. If P_4 does not broadcast anything or if $\mathbf{Y} \neq \{k^h_{w,z}\}_{h \in [3]}$, invoke simulator for $\mathsf{passive2PC}$, $\mathcal{S}_{\mathsf{passive2PC}}$ with $\mathcal{P}^2 = \mathcal{P} \setminus \{P_4^*, P_3\}$ to complete the simulation and send the output y to all on behalf of honest parties.

Figure 7.9: Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\mathsf{god4PC}}$ for $\mathsf{god4PC}$ with actively corrupt P_4^* and passively corrupt P_1°.

Security against actively corrupt P_4^* and passively corrupt P_1°: We now formally argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}^{\mathsf{4A,1P}}_{\mathsf{god4PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{god4PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_4 actively and P_1 passively. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{god4PC}, \mathcal{A}}$.

– HYB_1: Same as HYB_0 except: for share x^{23} (i.e. the share that the adversary doesn't get access to), replace c_{23} with the commitment of a dummy value in input commit phase. Do the same for share x^{32}.

– $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except: for wire w with share x^{23} and x^{32} (i.e. shares held only by the honest parties) assume a dummy value ($= 0$), compute $b_w = x_w \oplus \lambda_w$ using seed s_1.

– $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that some of the commitments of input keys sent by P_2, P_3 wrt seed s_1, which will not be opened are replaced with commitments of dummy values. These commitments correspond to the labels that do not correspond to any input share.

– $\textsc{hyb}_4$: Same as $\textsc{hyb}_3$ except: invoke $\mathcal{F}_{\mathsf{OT}}$ appropriately for the transfer of openings of key-shares corresponding to input wire w owned by garbler $P_g, g \in [3]$ and evaluator.

– $\textsc{hyb}_5$: Same as $\textsc{hyb}_4$ except: instead of constructing an honest GC, a simulated GC is constructed using the knowledge of seed s_1 (not known to the adversary), in such a way that each ciphertext for the output gate encrypts the same output key which corresponds to $b_w = y \oplus \lambda_w$ where y is obtained after having invoked $\mathcal{F}_{\mathsf{god}}$ and λ_w is known from the information of all seeds.

– $\textsc{hyb}_6$: Same as $\textsc{hyb}_5$ except: in $\textsc{hyb}_4$, $\mathbf{Y}$ is deemed to be invalid if there does not exist a bit z such that for each $j \in \mathcal{S}_g$, k_w^j obtained from $\mathbf{Y}$ matches $k_{w,z}^j$ while in $\textsc{hyb}_5$, it $\mathbf{Y}$ is deemed invalid if it is not the one that was encrypted in the simulated GC.

– $\textsc{hyb}_7$: Same as $\textsc{hyb}_6$ except: in case of a 2PC instance elected because of a public/private conflict, invoke simulator for $\mathsf{passive2PC}$ as in [LP04], instead of running $\mathsf{passive2PC}$.

Note that $\textsc{hyb}_7 = \textsc{ideal}_{\mathcal{F}_{\mathsf{god}}, \mathcal{S}_{\mathsf{god4PC}}^{\mathsf{4A,1P}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\textsc{hyb}_0 \overset{c}{\approx} \textsc{hyb}_1$: The only difference between the hybrids is that in $\textsc{hyb}_2$, the commitment for shares x^{32} and x^{23} are replaced by commitments of dummy values. Note that these are the shares whose openings are not revealed to the adversary. Hence, the indistinguishability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_1 \overset{c}{\approx} \textsc{hyb}_2$: The only difference in the value of b_w computed such that in $\textsc{hyb}_1$, it is w.r.t. honest share x_w while in $\textsc{hyb}_2$, it is w.r.t. dummy share 0. This remains indistinguishable to the adversary as she is unaware of seed s_1 and hence can't compute the underlying x_w.

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_3$: The only difference between the hybrids is that, in $\textsc{hyb}_3$, the commitments of input wire labels wrt seed s_1, which will not be opened are replaced with commitments on dummy values. The indistingushability follows from the hiding property of the commitment scheme.

$\textsc{hyb}_3 \overset{c}{\approx} \textsc{hyb}_4$: Indistinguishability of hybrids follows from reduction to the security of the underlying OT scheme [EGL85].

$\textsc{hyb}_4 \overset{c}{\approx} \textsc{hyb}_5$: Indistinguishability follows from reduction to the security of the underlying garbling scheme which breaks down to the security of PRF.

$\textsc{hyb}_5 \overset{c}{\approx} \textsc{hyb}_6$: Indistinguishability follows for the two different notions of validity of $\mathbf{Y}$ because a $\mathbf{Y}$ valid according to condition in $\textsc{hyb}_6$ is valid according to condition in $\textsc{hyb}_5$. Also a $\mathbf{Y}$ invalid according to condition in $\textsc{hyb}_6$ can possibly be valid according to condition in $\textsc{hyb}_5$ only if the adversary could forge the other output keys (i.e. which were not encrypted in simulated GC) for all the three seeds which is possible only with negligible probability according to the security of the garbling scheme.

$\textsc{hyb}_6 \overset{c}{\approx} \textsc{hyb}_7$: The indistinguishability follows from the indistinguishability of $\mathsf{passive2PC}$ simulator (follows from the security of [Yao82] presented in [LP04]) to the real execution of [Yao82]. $\square$

Chapter 8

4PC with Fairness

Relaxing the complexities in god4PC, we present an efficient constant round 4PC protocol that achieves fairness relying only on pairwise-private channels, against a mixed adversary that corrupts one party actively and the other passively. We give a quick overview highlighting the relaxations from god4PC, followed by challenges particular to the goal of fairness and the measures we take to tackle them. Note that, owing to the weaker security requirement of fairness, it is acceptable for the execution to abort before any party obtains the output.

8.1 The Construction

We retain the structure of four $\{P_1, P_2, P_3\}$ garblers and one evaluator P_4. The one-time SD is run as in god4PC. We let go of input distribution phase which was required in GOD protocol for the purpose of input consistency across executions in 4PC and 2PC. However, P_4 still distributes her input as additive shares among the garblers to employ the tricks of using only semi-honest OTs for key transfer as in god4PC. The garbling phase is run as in Fig 3.9 and every pair of garblers (as appointed by seed-distribution) send the GC fragment (common between them) to P_4, who checks the equality of two copies for correctness. Subsequently, P_4 evaluates the GC to obtain the encoded output $\mathbf{Y}$ and sends to the garblers for output construction. For the transfer of input mask-shares to the wire owner, since there exist two senders that can enforce the owner to abort if the values mismatch, we let go of commitments on input mask shares. The wire-owner computes the masked input and sends to P_4. For the wire w owned by a garbler P_g, she sends the keys corresponding to the two seeds she knows. To transfer of the third key k^g_{w,b_w} of the input super-key, two passive OTs are run as in god4PC. Note that, as robustness is not a requirement anymore, we strip off the commitments on input keys by the garblers done in god4PC. An incorrect key can at most lead to an invalid $\mathbf{Y}$ by the honest P_4. Hence, we enable the garblers to send hash of both output keys as part of the GC to P_4 who verifies for every wire, if all keys in the computed $\mathbf{Y}$ correspond to the same masked output bit (valid) or not.

During the output phase, there happens to be a trivial violation of fairness where a corrupt P_4 selectively sends the $\mathbf{Y}$ to garblers on obtaining the output herself. This issue occurs as the output mask shares are released by the garblers without any promise of output distribution by a possibly corrupt P_4. To tackle this, we ask the garblers to withhold the dispersal of shares to P_4 until a valid $\mathbf{Y}$ is received. This, however, shifts the power to a malicious garbler who can send an invalid mask-share leading to incorrect output. Both these cases are similar to the concerns described in fair5PC. Although the distribution of seeds ensures the existence of two senders for each share (one of which is honest/passive), however, the best that can be done is abort when the senders send mismatching copies. This still violates fairness, as the corrupt sender would have learnt the output. Hence, we require *commit-then-open* technique where, an agreement on the commitments to output mask-shares is made in the garbling phase which are opened only when a valid $\mathbf{Y}$ is received. Now, if the malicious sender sends faulty commitments in the offline phase, parties can simply abort. Else an agreement on commitments is made and the opening phase in the output phase is guaranteed to be robust. The SD also enforces the dependency of a malicious P_4 on at least one honest garbler to obtain the output and thus, rescues fairness to some extent. The only threat that still persists is selective distribution of $\mathbf{Y}$ by P_4 which we address by enforcing a garbler who received a valid $\mathbf{Y}$ from P_4 to further send the same to co-garblers. This ensures the following sequence of actions by a possibly corrupt P_4: either she does not send a valid $\mathbf{Y}$ to any honest party in which case she suffers, else she sends $\mathbf{Y}$ selectively, in which case our strategy ensures that everyone computes the output.

There exists subtle scenario where, despite an honest P_4 aborting during the circuit evaluation, a malicious garbler can convince the honest parties of any $\mathbf{Y}$ with the knowledge of all seeds (aided by a semi-honest co-garbler). This initiates the necessity of proof of origin of $\mathbf{Y}$ and is carried out similar to fair5PC. To elaborate, P_4 computes a collision resistant hash on a randomly chosen value in advance and the hash is agreed upon amongst the garblers. Consequently, in the output computation, P_4 sends the pre-image of the agreed upon hash along with $\mathbf{Y}$ as proof of origin of $\mathbf{Y}$. With this technique, an honest garbler receiving a valid $\mathbf{Y}$ along with a valid pre-image of the hash can be convinced that $\mathbf{Y}$ was indeed sent by P_4. The formal protocol appears in Fig 8.1.

All optimizations done in god4PC protocol can be adopted to fair4PC.

Protocol fair4PC

Input and Output Each party $P_i \in \mathcal{P}$ has x_i. Each party outputs $y = f(x_1, x_2, x_3, x_4)$ or $y = \perp$.

Common Inputs The circuit $C(x_1, x_2, x_3, \oplus_{j\in[3]} x^{4j})$ that takes the additive shares of P_4 as inputs and computes $f(x_1, x_2, x_3, x_4)$, each input, their shares and output are from $\{0,1\}$ (instead of $\{0,1\}^\ell$ for simplicity).

Notation $\mathcal{S}_g, g \in [3]$ denotes the indices of the seeds held by party P_g as well as the indices of parties who hold seed s_g.

Primitives A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ and eNICOM $(\mathsf{eCom}, \mathsf{eOpen})$, Oblivious Transfer (OT), Garble_3 (Fig 3.9), Eval_3 (Fig 3.10) and collision resistant hash H.

Seed Distribution (one-time): Parties P_1, P_2 and P_3 run π_{seedDist} (Fig 3.7).

Evaluator's Input Distribution: P_4 splits its input as $x_4 = x^{41} \oplus x^{42} \oplus x^{43}$ and sends x^{4g} to $P_g, g \in [3]$.

Proof of Origin Agreement: P_4 samples a random proof and computes $z = \mathsf{H}(\mathsf{proof})$. P_4 sends z to all the garblers who in turn exchange z and abort if all received copies of z are not the same.

Public Parameter for Equivocal Commitment: For eNICOM public parameter epp^g for $g \in [3]$, each $P_j, j \in [3] \setminus \{g\}$ samples epp^{gj} freshly (not derived from seeds) and sends to all. Each $P_i \in \mathcal{P}$ computes $\mathsf{epp}^g = \oplus_{j\in[3]\setminus\{g\}} \mathsf{epp}^{gj}$, forwards epp^g to all and aborts if any epp^gs received mismatch.

Equivocal commitment on output mask bits: For output wire w: $P_g, g \in [3]$ does the following for $j \in \mathcal{S}_g$:

– Computes $(\mathsf{c}_w^j, \mathsf{o}_w^j) \leftarrow \mathsf{eCom}(\mathsf{epp}^j, \lambda_w^j)$ and sends $(\mathsf{epp}^j, \mathsf{c}_w^j)$ to all. $P_i \in \mathcal{P}$ aborts if two mismatching copies of $(\mathsf{epp}^j, \mathsf{c}_w^j)$ are received.

Garbling Phase: Each garbler $P_g, g \in [3]$ runs $\mathsf{Garble}_3(C)$ (Fig 3.9) using $\mathcal{F}_{\mathsf{3AOT}}$ (Fig 3.8) instead of standard OT and sends $\{GC^j\}_{j\in\mathcal{S}_g}$ to P_4 who aborts if the copies of GC^j received mismatch. Else, P_4 sets $GC = GC^1||GC^2||GC^3$.

Input Phase: Let $\{k_{w,0}^j, k_{w,1}^j\}$ be the two keys derived from seed $\mathsf{s}_g, g \in [3]$ for input wire w.

– For input wire w owned by $P_g, g \in [3]$ having input bit x_g, each $P_j, j \in [3] \setminus \{g\}$ sends λ_w^g to P_g who aborts if the two copies of λ_w^g mismatch. Else, computes $\lambda_w = \oplus_{j\in[3]} \lambda_w^j$ and sets $b_w = x_g \oplus \lambda_w$. P_g sends $(b_w, k_{w,b_w}^j)_{j\in\mathcal{S}_g}$ to P_4. For key k_{w,b_w}^g corresponding to seed s_g that P_g does not possess, each $P_j, j \in [3] \setminus \{g\}$ additively shares the keys $k_{w,0}^g$ and $k_{w,1}^g$ as $k_{w,0}^g = [k_{w,0}^g]^0 \oplus [k_{w,0}^g]^1$ and $k_{w,1}^g = [k_{w,1}^g]^0 \oplus [k_{w,1}^g]^1$ (using randomness from s_g). Let $\{\alpha, \beta\} = [4] \setminus \{g, 4\}$. Further, the following is done:

- ∘ P_g runs a semi-honest OT acting as a receiver with choice bit b_w with P_α acting as sender with inputs $[k^g_{w,0}]^0, [k^g_{w,1}]^0$. Similarly, P_4 runs a semi-honest OT acting as a receiver with choice bit b_w with P_β acting as sender with inputs $[k^g_{w,0}]^1, [k^g_{w,1}]^1$. P_g receives $[k^g_{w,b_w}]^0$ as the OT output and sends to P_4 which is XORed by P_4 with his OT output i.e. $[k^g_{w,b_w}]^1$ to obtain k^g_{w,b_w}.

– For input wire w belonging to each of P_4's input share $x^{4l}, l \in [3]$, party $P_g, g \in [3]$ sends $\lambda^j_w, j \in \mathcal{S}_g$ to P_4 who aborts if the received copies of λ^j_w mismatch. Also, P_l receives λ^l_w from the other two garblers and aborts if the copies of λ^l_w mismatch. P_4, P_l compute $\lambda_w = \oplus_{j\in[3]}\lambda^j_w$ and set $b_w = x^{4l} \oplus \lambda_w$. For keys, a similar procedure as described in the previous step is done. Let $\mathbf{X}$ be the set of super-keys obtained for every input wire w i.e. $\{k^g_{w,b_w}\}_{g\in[3]}$.

Evaluation and Output Construction:

– P_4 runs Eval_3 and evaluates the DGC, GC using $\mathbf{X}$ to obtain the output super-key $\mathbf{Y} = \{k^g_w\}_{g\in[3]}$ and masked output $(y \oplus \lambda_w)$ for output wire w. P_4 computes $\mathsf{H}(k^g_w)$ and aborts if it not consistent with any hash received from the garblers as part of GC. Else, P_4 sends $Z = \{Y, \mathsf{proof}\}$ to all.

– Z sent by P_4 is deemed valid by P_g if both the following hold true: (i) there exists a bit b_w such that for each $j \in \mathcal{S}_g$, the k^j_w obtained from $\mathbf{Y}$ matches k^j_{w,b_w} (ii) $\mathsf{H}(\mathsf{proof}) = z$. If such a valid Z is received, $P_g, g \in [3]$ forwards Z to the co-garblers and $\mathsf{o}^j_w, j \in \mathcal{S}_g$ to all.

– A garbler P_α if received Z from a co-garbler but not from P_4 checks if Z is valid. If so, P_α sends $(\mathbf{Y}, \mathsf{proof}, \{\mathsf{o}^j_w\}_{j\in\mathcal{S}_\alpha})$ to co-garblers and $\{\mathsf{o}^j_w\}_{j\in\mathcal{S}_\alpha}$ to P_4.

– Each $P_i \in \mathcal{P}$ computes $\lambda_w = \oplus_{j\in[3]}\lambda^j_w$ using the mask shares obtained in the last two rounds (if sufficient) and obtains the output y by unmasking λ_w.

Figure 8.1: Protocol $\mathsf{fair4PC}$

We use equivocal commitment scheme to commit to the mask-shares on output wires for the same reason elaborated in Chapter 4. However, in the instantiation here, two parts of trapdoor are sufficient (as opposed to 4 in $\mathsf{fair5PC}$) to allow the simulator to learn the complete trapdoor while hiding it from the adversary in the real execution, owing to the existence of only one actively corrupt party.

8.2 Properties

Lemma 8.2.1. *The protocol* $\mathsf{fair4PC}$ *is correct.*

Proof. The input of P_4 is well defined by the shares sent to P_1, P_2, P_3. The 2 keys for each input wire owned by the garblers, along with the 3^{rd} key sent using OTs, define their committed inputs. Evaluation is done on committed inputs. The correctness of the keys received through OTs follows from the correctness of $\mathcal{F}_{\mathsf{OT}}$ [EGL85] along with the additive sharing of keys technique. The correctness of $\mathbf{Y}$ and thus y follows from the correctness of garbling and evaluation (Figs 3.9, 3.10). $\square$

Theorem 8.2.2. *The protocol* fair4PC *is securely realizes the functionality* $\mathcal{F}_{\mathsf{fair}}$ *(Fig 2.2) in the standard model against an adversary corrupting two parties–* 1 *active,* 1 *passive, assuming one-way permutations.*

The formal security proof is presented in Section 8.3.

We give the intuition of fairness for completeness. For fairness, we need to guarantee that if the adversary learns the output, then so do honest parties and converse. We first argue in the forward direction. Suppose an adversary gets the output. We consider two corruption cases: Firstly, when P_1 and P_4 are controlled by the adversary, the adversary obtains the output only if at least one honest garbler say P_2 receives a valid Z from P_4 or P_1 (valid shares of output wire mask bits also from P_1). If P_4 is passive, P_2 obtains Z directly from P_4 and sends the received message along with the masking bit shares she owns to all, allowing other parties to compute the output. The recipient garblers also send out their valid masking bit shares to all thus making all parties compute the output. When P_4 is active and P_2 receives valid Z from P_4, then P_2 sends Z and the openings on mask shares she holds to all. The recipient garblers also send out their valid masking bit shares to allow P_2 to compute the output. Else if P_4 is malicious and P_2 receives valid Z and openings from semi-honest P_1, P_2 computes the output and then sends the received message along with the openings of mask shares owned by P_2 to all, to allow each party to compute the output. Secondly, when two garblers P_1, P_2 are corrupt, an honest P_4 sends Z to all, on successfully evaluating GC. P_1, P_2, knowing all the seeds, can construct the output themselves. The honest garblers send the masking bit shares they hold to all. Thus, every party obtains the output in both cases.

To prove the converse case, suppose the honest parties get the output. We consider the same corruption cases as above. In the first case, it must be true that at least one of the honest garblers say P_2, received a valid Z who then sends the masking bit shares it owns along with Z to all. If P_2 received Z from P_4, then P_2 uses the masking bit shares sent by P_3 (once P_3 obtains output) to compute y. Else, P_2 must have received valid Z and the masking bit shares from P_1, which is sufficient to compute y. For the case of corrupt P_1, P_2, suppose P_4 gets the output. This implies that all garblers must have obtained the output using valid Z sent by P_4 and the masking bit shares received from co-garblers. Consequently, P_4 obtains the output using the masking bit shares sent by honest garblers. This summarizes the intuition.

8.3 Security Proof of fair4PC

We now outline the complete security proof of Theorem 8.2.2 that describes the security of the fair4PC protocol relative to its ideal functionality in the standard security model.

Proof. We describe the simulator $\mathcal{S}_{\mathsf{fair4PC}}$ for three cases which exhaustively cover the corruption scenarios: First, when P_1 is actively corrupt and P_2 is passively corrupt. Second, when P_1 is actively corrupt and P_4 is passively corrupt. Finally, when P_4 is actively corrupt and P_1 is passively corrupt. The corruption of any two garblers is symmetric to the case when P_1, P_2 are corrupt, the corruption of any one actively corrupt garbler and passively corrupt evaluator is symmetric to the second case and the corruption of any one passively corrupt garbler and actively corrupt evaluator is symmetric to the third case. The simulator acts on behalf of all honest parties in the execution. For better understanding we separate out the simulation for the subroutine π_{seedDist} from the simulation of main protocol in the $\mathcal{F}_{\mathsf{OT}}$ hybrid model.

We briefly highlight the need for equivocal commitment scheme (eNICOM) for the shares of output masking bits in our fair protocol as follows: The adversary can decide to abort the execution as late as when $\mathbf{Y}$ needs to be sent (in the worst case). Consequently, this enforces the simulator to make this decision on behalf of the adversary at the end of evaluation phase when calling the functionality. Hence, the simulator needs a mechanism to simulate the earlier rounds appropriately such as sending the GC and committing to the shares of the output masking bits, without the knowledge of whether the execution will result in a valid output or not (with no information about the output). The sending of distributed GC is handled as in any standard distributed garbling proof. To tackle the commitment on shares of output masking bits, the simulator commits to dummy bits for the seed completely under its control. At a later point if the execution results in invoking $\mathcal{F}_{\mathsf{fair}}$ and obtaining y, the simulator equivocates the commitments to desired share bits such that for each output wire w, $y \oplus \lambda_w$ decodes to correct y. The trapdoor and public parameter for our eNICOM scheme are derived from relevant seeds as described in the protocol.

We provide a high level view of the simulation in distributed garbling and evaluation for completeness. First, in the case of P_1^* actively corrupt and P_2° passively corrupt, the evaluator P_4 is honest. Hence correctness is required from the DGC. The simulator behaves as an honest P_3 following the protocol steps and instructing the functionality to abort in case of any cheating throughout the garbling since all seeds are known to the adversary. If no cheating is detected throughout the DGC construction, then the GC is generated as per the Garble_3 procedure. The inputs of corrupt parties are extracted during the garbled input communication. The simulator sends abort to the functionality if the GC partition sent by P_1^* is not same as the one generated by honest parties.

Second, in the case of actively corrupt P_1^* and passively corrupt P_4°, the simulator knows the seeds held by the adversary. In addition the simulator has complete control over the part of GC generated using seed s_1. Since the simulator does not know the output in advance, the masking bit share λ_w^1 corresponding to output wires w cannot be set in advance. As a result, a fake GC is constructed using s_1 that always evaluates to the same output super-key for the extracted and random inputs that are known to the simulator. If the evaluation goes through and $\mathbf{Y}$ is received on behalf of the honest parties, then the simulator invokes the functionality to obtain y, aptly programs the masking bit share under its control by setting $\lambda_w^1 = y \oplus (\oplus_{i\in[3]}, i \neq 1)\lambda_w^i$ for each output wire, performs equivocation on the commitment made for share λ_w^1 and sends the corresponding decommitment to the corrupt parties thus completing simulation. A similar strategy as explained in the second case is employed for the case when P_1° is passively corrupt and P_4^* is actively corrupt We describe the simulator steps in detail for π_{seedDist} and the main protocol separately in Figs 8.2, 8.4, 8.6 and 8.3, 8.5, 8.7 respectively.

Simulator $\mathcal{S}_{\pi_{\mathsf{seedDist}}}^{\mathsf{1A,2P}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1^* as sender and P_3 as receiver to obtain seed s_2.

- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2° as receiver.

- For the commitment instance between P_1^* as sender and P_2° as receiver to commit to seed s_3:

  - Run the ExtCom protocol where P_1^* and P_2° run rounds 1-3 and broadcast their messages $(\mathsf{extcom}_1^1, \mathsf{extcom}_2^1, \mathsf{extcom}_3^1)$.
  - Rewind the adversary to the end of round 1 for P_1^* and P_2° to rerun rounds 2-3 and broadcast $(\mathsf{extcom}_2^2, \mathsf{extcom}_3^2)$.
  - On behalf of P_3, Run extractor algorithm $\mathsf{Extract}$ of the commitment scheme as in Fig 2.4 using inputs $(\mathsf{extcom}_1^1, \{\mathsf{extcom}_2^i, \mathsf{extcom}_3^i\}_{i\in[2]})$ to extract the committed seed s_3.

Figure 8.2: Simulator $\mathcal{S}_{\pi_{\mathsf{seedDist}}}^{\mathsf{1A,2P}}$ for π_{seedDist} with actively corrupt P_1^* and passively corrupt P_2°

Protocol $\mathcal{S}_{\mathsf{fair4PC}}^{\mathsf{1A,2P}}$

Seed Distribution Phase (one-time): Invoke $\mathcal{S}_{\pi_{\mathsf{seedDist}}}^{\mathsf{1A,2P}}$ (Fig 8.2). Extract s_3.

Evaluator's Input Distribution: Sample random x^{41}, x^{42} and send $x^{4g}, g \in [2]$ to P_g on behalf of P_4.

Proof of Origin Agreement: On behalf of P_4: sample a random proof and compute $z = \mathsf{H}(\mathsf{proof})$. Send z to P_1^*, P_2°. In the next round, on behalf of P_3: receive z from P_1^*, P_2° and send z to P_1^*, P_2°. If P_1^* sent a different value of z from what was computed by simulator, invoke $\mathcal{F}_{\mathsf{fair}}$ (Fig 2.2) with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Public parameter of equivocal commitment:

- For eNICOM public parameter epp^g for $g \in [2]$: On behalf of P_3, sample epp^{g3} using fresh randomness (not derived from seeds) and send to P_1^*, P_2°. On behalf of P_3, P_4 receive $\mathsf{epp}^{gh}, h \in [2] \setminus \{g\}$ from P_h, compute $\mathsf{epp}^g = \mathsf{epp}^{g3} \oplus \mathsf{epp}^{gh}$, send (and receive) epp^g to (from) P_1^*, P_2° and receive epp^g from P_1^*, P_2°. If a different value of epp^g received from P_1^*, P_2°, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^*, P_2° and set $y = \perp$.
- For eNICOM public parameter epp^3: On behalf of P_3, P_4, receive $\mathsf{epp}^{3g}, g \in [2]$ from P_g. Compute $\mathsf{epp}^3 = \mathsf{epp}^{31} \oplus \mathsf{epp}^{32}$ send (and receive) epp^3 to (from) P_1^*, P_2° (on behalf of P_4). If a different value of epp^g received from P_1^*, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Equivocal commitment on output mask bits: Do the following for output wire w:

- On behalf of P_3 and $j \in \mathcal{S}_3$, compute $(\mathsf{c}_w^j, \mathsf{o}_w^j) \leftarrow \mathsf{eCom}(\mathsf{epp}^j, \lambda_w^j)$ and send $(\mathsf{epp}^j, \mathsf{c}_w^j)$ to all. If two mismatching copies of $(\mathsf{epp}^j, \mathsf{c}_w^j), j \in \mathcal{S}_1$ are received (due to misbehaviour by P_1^*) on behalf of $P_i, i \in \{3, 4\}$, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Garbling Phase: On behalf of P_3: Run Garble_3 honestly with $\mathcal{F}_{\mathsf{3AOT}}$ (Fig 3.8) as means to achieve OT using $\mathsf{s}_1, \mathsf{s}_2$. On behalf of P_4, receive GC^j from the corrupt garblers. If two mismatching copies of $GC^j, j \in \mathcal{S}_1$ are received (due to misbehaviour by P_1^*), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. Else, set $GC = GC^1 || GC^2 || GC^3$.

Input and Key Transfer: Let $\{k_{w,0}^j, k_{w,1}^j\}$ be the two keys derived for wire w from seed $\mathsf{s}_j, j \in [3]$.

- For input wire w belonging to P_1^* having input bits x_1: on behalf of P_3, send λ_w^1 to P_1. Similar steps are done for x_2 of P_2°. Receive λ_w^3 from P_1^* and P_2° for input wire w belonging to P_3 having input bit x_3. If they send mismatching copies of λ_w^3, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. Else, compute $\lambda_w = \oplus_{j \in [3]} \lambda_w^j$ and set $b_w = \lambda_w$ (assuming a dummy value of $x_3 = 0$).
- On behalf of P_4 and input wire w belonging to $P_g, g \in [2]$, receive $(b_w, k_{w,b_w}^j)_{j \in \mathcal{S}_g}$. For key k_{w,b_w}^g corresponding to seed s_g that P_g does not possess, on behalf of P_3: split the keys $k_{w,0}^g$ and $k_{w,1}^g$ as $k_{w,0}^g = [k_{w,0}^g]^0 \oplus [k_{w,0}^g]^1$ and $k_{w,1}^g = [k_{w,1}^g]^0 \oplus [k_{w,1}^g]^1$ using randomness from s_g. Further, the following is done:

◦ Receive $[k^g_{w,b_w}]^0$ (obtained by P_g via OT run with the corrupt co-garbler) on behalf of P_4 from P_g.

- For input wire w belonging to P_3: Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_3 (as receiver) and P_1^* (as sender). Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4 (as receiver) and P_2° (as sender).
- For input wire w corresponding to each of P_4's input share $x^{4l}, l \in [3]$, on behalf of P_4: receive $\lambda^j_w, j \in \mathcal{S}_g$ from $P_g, g \in [2]$. On behalf of P_3, receive λ^3_w from P_1^* and P_2° (for wire corresponding to share x^{43}) and send λ^g_w to $P_g, g \in [2]$ (for wire corresponding to share x^{4g}). If mismatching copies are received (because of misbehaviour by P_1^*), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^*, P_2° and set $y = \perp$. For the keys, a similar procedure as described in the previous step is done.

Evaluation and Output Construction:

- Let $\tilde{\mathbf{X}}$ be the set of super-keys obtained (w.r.t. the inputs of the adversary and the dummy input values assumed for the honest parties). Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_2)$ on behalf of corrupt P_1^*, P_2° and obtain y. From the knowledge of all seeds, compute $b_w = y \oplus \lambda_w$ and $\mathbf{Y} = \{k^g_{w,b_w}\}_{g \in [3]}$. On behalf of P_4, send $Z = \{Y, \mathsf{proof}\}$ to P_1^*, P_2°.
- On behalf of P_4, receive o^j_w for $j \in \mathcal{S}_g$ from $P_g, g \in [2]$. On behalf of P_3, receive (Z, o^j_w) for $j \in \mathcal{S}_g$ from P_g and send $(Z, o\mathsf{o}^j_w)$ for $j \in \mathcal{S}_3$ to P_1^*, P_2°.

Figure 8.3: Simulator $\mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{fair4PC}}$ for $\mathsf{fair4PC}$ with actively corrupt P_1^* and passively corrupt P_2°.

The hybrid arguments are as defined below.

Security against actively corrupt P_1^* and passively corrupt P_2°: We now formally argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{fair4PC}}} \overset{c}{\approx} \textsc{real}_{\mathsf{fair4PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1 actively and P_2 passively. The views are shown to be indistinguishable via a series of intermediate hybrids.

– $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{fair4PC}, \mathcal{A}}$.

– $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except: rerun rounds 2-3 of extractable commitment (with P_2 as sender and P_1^* as receiver) in the seed-distribution phase to extract seed s_3. Run the subsequent rounds same as $\textsc{hyb}_0$.

– $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except: for input wire w held by garbler (say P_3) and P_4°, to obtain opening k^3_{w,b_w}, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_3 as receiver and P_1^* as sender to obtain $[k^3_{w,b_w}]^0$. Similarly, invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4° as receiver and P_2° as sender to obtain $[k^3_{w,b_w}]^1$.

– $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except: if the construction of distributed GC fails or $\mathbf{X}$ is not obtained, output $\perp$ on behalf of P_3.

– $\textsc{hyb}_4$: Same as $\textsc{hyb}_3$ except: if evaluation of distributed GC proceeds, compute $z = y \oplus \lambda_w$ (where y is the output obtained on invoking $\mathcal{F}_{\mathsf{fair}}$ and λ_w is the mask computed from the knowledge of all seeds) and set $\mathbf{Y} = \{k^g_{w,z}\}_{g\in[3]}$ and send $\mathbf{Y}$ to the adversary parties (instead of $\mathbf{Y}$ obtained from the evaluation of distributed GC).

Note that $\textsc{hyb}_4 = \textsc{ideal}_{\mathcal{F}_{\mathsf{fair}},\mathcal{S}^{\mathsf{1A,2P}}_{\mathsf{fair4PC}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\textsc{hyb}_0 \overset{c}{\approx} \textsc{hyb}_1$: The only difference between the hybrids is that, in $\textsc{hyb}_1$, rounds 2-3 of the extractable commitment are rewound. However, the adversary's view contains only the final rewound execution and the previous rewinds are erased. Hence the hybrids $\textsc{hyb}_0$ and $\textsc{hyb}_1$ are indistinguishable.

$\textsc{hyb}_1 \overset{c}{\approx} \textsc{hyb}_2$: Indistinguishability of hybrids follows from the security of the underlying OT scheme [EGL85].

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_3$: In $\textsc{hyb}_2$, P_3 could have obtained a non-$\perp$ value for y even though P_4° failed in GC evaluation by if it received a valid $Z = (\mathbf{Y}, \mathsf{proof})$ from active P_1^* such that $\mathbf{Y}$ is valid and $z = \mathsf{H}(\mathsf{proof})$. P_1 can forge a valid $\mathbf{Y}$ because of the knowledge of all seeds. This can be reduced to the pre-image resistant property of the hash function according to which P_1^* could forge a pre-image of z to come up with a valid value of proof only with negligible probability.

$\textsc{hyb}_3 \overset{c}{\approx} \textsc{hyb}_4$: The indistinguishability follows from the correctness of the garbling scheme (follows from Lemma 3.1.4) since $\mathbf{Y}$ computed using the Evaluation Phase of garbling would also result in $\mathbf{Y} = \{k^g_{w,y\oplus\lambda_w}\}_{g\in[3]}$ where $y = f(x_1, x_2, x_3, x_4)$.

Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\pi_{\mathsf{seedDist}}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1^* as sender and P_3 as receiver to obtain seed s_2. Abort if P_1^* sends incorrect opening.
- Sample random s_3 and act honestly on behalf of P_2 for the commitment instance between P_2 as sender and P_1^* as receiver.
- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2 as receiver.

Figure 8.4: Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\pi_{\mathsf{seedDist}}}$ for π_{seedDist} with actively corrupt P_1^* and passively corrupt P_4°

Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\mathsf{fair4PC}}$

Seed Distribution Phase (one-time): Invoke $\mathcal{S}^{\mathsf{1A,4P}}_{\pi_{\mathsf{seedDist}}}$ (Fig 8.2).

Evaluator's Input Distribution: On behalf of $P_g, g \in \{2,3\}$, receive x^{4g} from P_4°.

Proof of Origin Agreement: On behalf of $P_g, g \in \{2,3\}$: receive z from P_4°. In the next round, send z to P_1^* and receive z from P_1^*. If P_1^* sends a different value of z from what was received from P_4°, invoke $\mathcal{F}_{\mathsf{fair}}$ (Fig 2.2) with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Public parameter of equivocal commitment:

- For eNICOM public parameter epp^1: On behalf of P_2, P_3, sample $\mathsf{epp}^{12}, \mathsf{epp}^{13}$ using fresh randomness (not derived from seeds) and send to P_1^*. On behalf of P_2, P_3 receive epp^1 from P_1^*. If a different value of epp^1 received from P_1^* on behalf of honest parties, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

- For eNICOM public parameter $\mathsf{epp}^g, g \in \{2,3\}$: On behalf of P_g, receive epp^{g1} from P_1^*. Compute epp^g. Then, send (and receive) epp^g to (from) P_1^*. If a different value of epp^g received from P_1^* (on behalf of honest $P_h, h \neq g$), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Equivocal commitment on output mask bits: Do the following for output wire w:

- On behalf of $P_g, g \in \{2,3\}$ and $j \in \mathcal{S}_g$, compute $(\mathsf{c}_w^j, \mathsf{o}_w^j) \leftarrow \mathsf{eCom}(\mathsf{epp}^j, \lambda_w^j)$ and send $(\mathsf{epp}^j, \mathsf{c}_w^j)$ to all. If P_1^* sends a different copy of $(\mathsf{epp}^j, \mathsf{c}_w^j)$ for $j \in \mathcal{S}_1$ from what was computed on behalf of the honest parties, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Garbling Phase: On behalf of $P_g, g \in \{2,3\}$: Run Garble_3 using $\mathcal{F}_{\mathsf{3AOT}}$ (Fig 3.8) as means to achieve OT honestly using the knowledge of all seeds such that each ciphertext for the output gate of GC^g for $g \in [3]$ encrypts the same output key $k^g_{w,z}$, for $z \in \{0,1\}$. Send $\{GC^j\}$ for $j \in \mathcal{S}_g$. If P_4° aborts (due to misbehavior by P_1^*), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. Else, set $GC = GC^1 || GC^2 || GC^3$.

Input and Key Transfer: Let $\{k^j_{w,0}, k^j_{w,1}\}$ be the two keys derived for wire w from seed $\mathsf{s}_j, j \in [3]$.

- For input wire w belonging to P_1^* having input bit x_1: on behalf of $P_g, g \in \{2,3\}$, send λ_w^1 to P_1^*. Receive λ_w^g from P_1^* for input wire w belonging to P_g having input bit x_g. If P_1^* sends an incorrect value (which can be checked based on the knowledge of s_g), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. Else, compute $\lambda_w = \oplus_{j \in [3]} \lambda_w^j$ and set $b_w = \lambda_w$ (assuming a dummy value of $x_g = 0$).

- On behalf of $P_g, g \in \{2,3\}$ and input wire w belonging to P_g, send $(b_w, k^j_{w,b_w})_{j \in \mathcal{S}_g}$ to P_4°. For key k^g_{w,b_w} corresponding to seed s_g that P_g does not possess, on behalf of $P_h, h \in [3] \setminus \{1,g\}$: split the keys $k^g_{w,0}$ and $k^g_{w,1}$ as $k^g_{w,0} = [k^g_{w,0}]^0 \oplus [k^g_{w,0}]^1$ and $k^g_{w,1} = [k^g_{w,1}]^0 \oplus [k^g_{w,1}]^1$ using randomness from s_g. Further, the following is done:
  - Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_g (as receiver) and P_1^* (as sender). Invoke another $\mathcal{F}_{\mathsf{OT}}$ with P_4° (as receiver) and P_h (as sender).
  - Send $[k^g_{w,b_w}]^0$ on behalf of P_g to P_4°.
- For input wire w belonging to P_1^* corresponding to input x_1: Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_1^* (as receiver) and P_2 (as sender). Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4° (as receiver) and P_3 (as sender) and receive b_w sent by P_4° to $\mathcal{F}_{\mathsf{OT}}$. Compute $x_1 = b_w \oplus \lambda_w$ (from the knowledge of all seeds).
- For input wire w corresponding to each of P_4°'s input share $x^{4l}, l \in [3]$: on behalf of $P_g, g \in \{2,3\}$, send λ^j_w for $j \in \mathcal{S}_g$ to P_4°. If P_4° aborts (due to misbehavior by P_1^*), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.
- For input wire w corresponding to share $x^{4g}, g \in \{2,3\}$, on behalf of P_g: receive λ^g_w from P_1^*. If P_1^* sends different value of λ^g_w from what was computed by the simulator (using knowledge of s_g), invoke $\mathcal{F}_{\mathsf{fair}}$ (Fig 2.2) with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. For input wire w corresponding to share x^{41}, send λ^1_w on behalf of P_2, P_3 to P_1^*. If P_1^* aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. For the keys, a similar procedure as described in the previous step is done to compute x^{41}.

Evaluation and Output Construction:

- If P_4° aborts (due to unsuccessful evaluation), invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$. Else, receive $Z = \{Y, \mathsf{proof}\}$ from P_4° on behalf of $P_g, g \in \{2,3\}$: compute b_w such that k^j_w obtained from $\mathbf{Y}$ matches with k^j_{w,b_w} for $j \in [3]$. Invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_4)$ on behalf of corrupt P_1^*, P_4° and obtain y.
- Set $\lambda^1_w = y \oplus \oplus_{g \in [3], g \neq 1} \lambda^1_w$. Run $\mathsf{Equiv}(\mathsf{c}^1_w, \mathsf{o'}^1_w, \lambda^1_w, t)$ (where t is the trapdoor corresponding to epp^1) to obtain o^1_w which opens $c\mathsf{c}^1_w$ to λ^1_w and send o^1_w to P_4° and (Z, o^1_w) to P_1° on behalf of P_g. Receive (Z, o^j_w) from P_1° for $j \in \mathcal{S}_1$ on behalf of P_g.

Figure 8.5: Simulator $\mathcal{S}^{\mathsf{1A,4P}}_{\mathsf{fair4PC}}$ for $\mathsf{fair4PC}$ with actively corrupt P_1^* and passively corrupt P_4°

Security against actively corrupt P_1^* and passively corrupt P_4°: We now formally argue that $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}^{\mathsf{1A,4P}}_{\mathsf{fair4PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{fair4PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1 actively and P_4 passively. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{fair4PC}, \mathcal{A}}$.

– HYB_1: Same as HYB_0 except: For wire w belonging to $P_g, g \in \{2,3\}$ with input bit x_g, assume a dummy value of $x_g = 0$.

– $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except: Invoke $\mathcal{F}_{\mathsf{OT}}$ appropriately for the transfer of openings of key-shares corresponding to each input wire w.

– $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that,

- $\textsc{hyb}_{3.1}$: When the execution results in `abort`, the commitment to λ_w^1 for output wire w is created for a dummy value.
- $\textsc{hyb}_{3.2}$: When the execution results in output y, the commitment c_w^1 for each output wire w is created for a dummy value and later equivocated to λ_w^1 using o_w^1 computed via where $\mathsf{o}_w^1 = \mathsf{Equiv}(\mathsf{c}_w^1, \mathsf{o}_w'^1, \lambda_w^1, t)$ where t is the trapdoor for the commitment c_w^1.

– $\textsc{hyb}_4$: Same as $\textsc{hyb}_3$ except that that the protocol results in abort if the received $\mathbf{Y}$ does not correspond to the $\mathbf{Y}$ resulting from the simulated GC.

Note that $\textsc{hyb}_4 = \textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{fair4PC}}^{\mathsf{1A,4P}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\textsc{hyb}_0 \overset{c}{\approx} \textsc{hyb}_1$: The only difference is in the value of b_w computed such that in $\textsc{hyb}_0$, it is w.r.t. honest share x_g for $g \in \{2, 3\}$ while in $\textsc{hyb}_1$, it is w.r.t. dummy value 0. This remains indistinguishable to the adversary because he is unaware of seed s_1 and hence can't compute the underlying x_w.

$\textsc{hyb}_1 \overset{c}{\approx} \textsc{hyb}_2$: Indistinguishability of hybrids follows from reduction to the security of the underlying OT [EGL85].

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_{3.1}$: The difference between the hybrids is that the commitment to λ_w^1 for each output wire w, is created for a dummy value in $\textsc{hyb}_{3.1}$. The indistinguishability follows via reduction to the hiding property of eCom.

$\textsc{hyb}_2 \overset{c}{\approx} \textsc{hyb}_{3.2}$: The difference between the hybrids is that in $\textsc{hyb}_{3.2}$, commitment to λ_w^1 for each output wire w, is created for a dummy value and later equivocated using o_w^1 computed via where $\mathsf{o}_w^1 = \mathsf{Equiv}(\mathsf{c}_w^1, \mathsf{o}_w'^1, \lambda_w^1, t)$ where t is the trapdoor for the commitment c_w^1. Indistinguishability follows via reduction to the hiding property of eCom.

$\textsc{hyb}_3 \overset{c}{\approx} \textsc{hyb}_4$: The only difference between the hybrids is that, in $\textsc{hyb}_3$, the protocol aborts if for some output wire w and index $j \in \mathcal{S}_g$, k_{w,b_w}^j of the received $\mathbf{Y}$ does not match with either $(k_{w,0}^j, k_{w,1}^j)$ or the keys $\{k_{w,b_w}^j\}_{j \in \mathcal{S}_g}$ in $\mathbf{Y}$ do not map to the same b_w whereas in $\textsc{hyb}_4$, the protocol results in abort if the received $\mathbf{Y}$ does not match the one created with simulated GC. By security of the garbling scheme, P_4 could have forged such a $\mathbf{Y}$ only with negligible probability.

Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\pi_{\mathsf{seedDist}}}$

- Act honestly on behalf of P_3 for the commitment instance between P_1° as sender and P_3 as receiver to obtain seed s_2.
- Sample random s_3 and act honestly on behalf of P_2 for the commitment instance between P_2 as sender and P_1° as receiver.
- Sample random s_1 and act honestly on behalf of P_3 for the commitment instance between P_3 as sender and P_2 as receiver.

Figure 8.6: Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\pi_{\mathsf{seedDist}}}$ for π_{seedDist} with actively corrupt P_4^* and passively corrupt P_1°

Simulator $\mathcal{S}^{\mathsf{4A,1P}}_{\mathsf{fair4PC}}$

Seed Distribution Phase (one-time): Invoke $\mathcal{S}^{\mathsf{4A,1P}}_{\pi_{\mathsf{seedDist}}}$ (Fig 8.2).

Evaluator's Input Distribution: On behalf of $P_g, g \in \{2,3\}$, receive x^{4g} from P_4^*.

Proof of Origin Establishment: On behalf of $P_g, g \in \{2,3\}$: receive z from P_4^*. If P_4^* sends different values of z to P_2 and P_3, invoke $\mathcal{F}_{\mathsf{fair}}$ (Fig 2.2) with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_4^* and set $y = \perp$. Else, in the next round, send z to P_1° and receive z from P_1°. If P_1 sends a different value of z from what was received from P_4^*, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_4^* and set $y = \perp$.

Public parameter of equivocal commitment:

- For eNICOM public parameter epp^1: On behalf of P_2, P_3, sample $\mathsf{epp}^{12}, \mathsf{epp}^{13}$ using fresh randomness (not derived from seeds) and send to P_1°, P_4^*. On behalf of P_2, P_3 receive epp^1 from P_4^*. If a different value of epp^1 received from P_4^* on behalf of honest parties, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_4^* and set $y = \perp$.
- For eNICOM public parameter $\mathsf{epp}^g, g \in \{2,3\}$: On behalf of P_g, receive epp^{g1} from P_1°. Compute epp^g. Then, send (and receive) epp^g to (from) P_1°. If a different value of epp^g received from P_4^* on behalf of honest parties, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_4^* and set $y = \perp$.

Equivocal commitment on output mask bits: Do the following for output wire w:

- On behalf of $P_g, g \in \{2,3\}$ and $j \in \mathcal{S}_g$, compute $(\mathsf{c}_w^j, \mathsf{o}_w^j) \leftarrow \mathsf{eCom}(\mathsf{epp}^j, \lambda_w^j)$ and send $(\mathsf{epp}^j, \mathsf{c}_w^j)$ to all. Receive $(\mathsf{epp}^j, \mathsf{c}_w^j), j \in \mathcal{S}_1$ from P_1° on behalf of honest parties.

Garbling Phase: On behalf of $P_g, g \in \{2,3\}$: Run Garble_3 using $\mathcal{F}_{\mathsf{3AOT}}$ (Fig 3.8) as means to achieve OT honestly using the knowledge of all seeds such that each ciphertext for the output gate of GC^g for $g \in [3]$ encrypts the same output key $k_{w,z}^g$ for $z \in \{0,1\}$. Send $\{GC^j\}$ for $j \in \mathcal{S}_g$. If P_4^* aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_4^* and set $y = \perp$.

Input and Key Transfer: Let $\{k_{w,0}^j, k_{w,1}^j\}$ be the two keys derived for wire w from seed $\mathsf{s}_j, j \in [3]$.

- For input wire w belonging to P_1° having input bit x_1: on behalf of $P_g, g \in \{2,3\}$, send λ_w^1 to P_1°. Receive λ_w^g from P_1° for input wire w belonging to P_g having input bit x_g. Compute $\lambda_w = \oplus_{j \in [3]} \lambda_w^j$ and set $b_w = \lambda_w$ (assuming a dummy value of $x_g = 0$).
- On behalf of $P_g, g \in \{2,3\}$ and input wire w belonging to P_g, send $(b_w, k_{w,b_w}^j)_{j \in \mathcal{S}_g}$ to P_4. For key k_{w,b_w}^g corresponding to seed s_g that P_g does not possess, on behalf of $P_h, h \in [3] \setminus \{1, g\}$: split the keys $k_{w,0}^g$ and $k_{w,1}^g$ as $k_{w,0}^g = [k_{w,0}^g]^0 \oplus [k_{w,0}^g]^1$ and $k_{w,1}^g = [k_{w,1}^g]^0 \oplus [k_{w,1}^g]^1$ using randomness from s_g. Further, the following is done:
  - Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_g (as receiver) and P_1° (as sender). Invoke another $\mathcal{F}_{\mathsf{OT}}$ with P_4^* (as receiver) and P_h (as sender).
  - Send $[k_{w,b_w}^g]_a$ on behalf of P_g to P_4^*.
- For input wire w corresponding to each of P_4^*'s input share $x^{4l}, l \in [3]$: on behalf of $P_g, g \in \{2,3\}$, send λ_w^j for $j \in \mathcal{S}_g$ to P_4^*. If P_4^* aborts, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_4^* and set $y = \perp$.
- For input wire w corresponding to share $x^{4g}, g \in \{2,3\}$, on behalf of P_g: receive λ_w^g from P_1°.
- For input wire w belonging to P_4^* corresponding to input x^{41}: Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_1° (as receiver) and P_2 (as sender) and receive b_w sent by P_1° to $\mathcal{F}_{\mathsf{OT}}$. Compute $x_1 = b_w \oplus \lambda_w$ (from the knowledge of all seeds). Invoke $\mathcal{F}_{\mathsf{OT}}$ with P_4^* (as receiver) and P_3 (as sender). Similar steps are done for x_1.

Evaluation and Output Construction:

- On behalf of $P_g, g \in \{2,3\}$: receive $Z = \{\mathbf{Y}, \mathsf{proof}\}$ from P_4^*. If P_g receives a valid Z, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_4)$ on behalf of corrupt P_1°, P_4^* and obtain y. Set $\mathsf{o}_w^1 = \mathsf{Equiv}(\mathsf{c}_w^1, \mathsf{o'}_w^1, \lambda_w^1, t)$ where t is the trapdoor corresponding to epp^1 and send o_w^j to P_4^* and (Z, o_w^j) for $j \in \mathcal{S}_g$ to P_1° on behalf of P_g. Receive (Z, o_w^j) from P_1° for $j \in \mathcal{S}_1$ on behalf of P_g.
- On behalf of $P_g, g \in \{2,3\}$: If neither P_g receives valid Z from P_4^* but a valid Z, o_w^j for $j \in \mathcal{S}_1$ is received in the subsequent round from P_1°, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, x_1), (\mathsf{Input}, x_4)$ on behalf of corrupt P_1°, P_4^* and obtain y. Set $\mathsf{o}_w^1 = \mathsf{Equiv}(\mathsf{c}_w^1, \mathsf{o'}_w^1, \lambda_w^1, t)$ where t is the trapdoor corresponding

to epp^1 and send o_w^j to P_4 and (Z, o_w^j) for $j \in \mathcal{S}_g$ to P_1° on behalf of P_g.

- If neither P_g for $g \in \{2,3\}$ receives valid Z from P_4 or from P_1 in the subsequent round, invoke $\mathcal{F}_{\mathsf{fair}}$ with $(\mathsf{Input}, \perp)$ on behalf of corrupt P_1^* and set $y = \perp$.

Figure 8.7: Simulator $\mathcal{S}_{\mathsf{fair4PC}}^{\mathsf{4A,1P}}$ for $\mathsf{fair4PC}$ with actively corrupt P_4^* and passively corrupt P_1°

Security against actively corrupt P_4^* and passively corrupt P_1°: We now formally argue that $\textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{fair4PC}}^{\mathsf{4A,1P}}} \overset{c}{\approx} \textsc{real}_{\mathsf{fair4PC}, \mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_4 actively and P_1 passively. The views are shown to be indistinguishable via a series of intermediate hybrids.

- $\textsc{hyb}_0$: Same as $\textsc{real}_{\mathsf{fair4PC}, \mathcal{A}}$.
- $\textsc{hyb}_1$: Same as $\textsc{hyb}_0$ except: For wire w belonging to $P_g, g \in \{2,3\}$ with input bit x_g, assume a dummy value of $x_g = 0$.
- $\textsc{hyb}_2$: Same as $\textsc{hyb}_1$ except: Invoke $\mathcal{F}_{\mathsf{OT}}$ appropriately for the transfer of openings of key-shares corresponding to each input wire w.
- $\textsc{hyb}_3$: Same as $\textsc{hyb}_2$ except that,
  - $\textsc{hyb}_{3.1}$: When the execution results in `abort`, the commitment to λ_w^1 for output wire w is created for a dummy value.
  - $\textsc{hyb}_{3.2}$: When the execution results in output y, the commitment c_w^1 for each output wire w is created for a dummy value and later equivocated to λ_w^1 using o_w^1 computed via where $\mathsf{o}_w^1 = \mathsf{Equiv}(\mathsf{c}_w^1, \mathsf{o}_w'^1, \lambda_w^1, t)$ where t is the trapdoor for the commitment c_w^1.
- $\textsc{hyb}_4$: Same as $\textsc{hyb}_3$ except that that the protocol results in abort if the received $\mathbf{Y}$ does not correspond to the $\mathbf{Y}$ resulting from the simulated GC.

Note that $\textsc{hyb}_3 = \textsc{ideal}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{fair4PC}}^{\mathsf{4A,1P}}}$. Next, we show that each pair of hybrids is computationally indistinguishable as follows:

$\textsc{hyb}_0 \overset{c}{\approx} \textsc{hyb}_1$: The only difference is in the value of b_w computed such that in $\textsc{hyb}_0$, it is w.r.t. honest share x_g for $g \in \{2,3\}$ while in $\textsc{hyb}_1$, it is w.r.t. dummy value 0. This remains indistinguishable to the adversary because she is unaware of seed s_1 and hence can't compute the underlying x_w.

$\textsc{hyb}_1 \overset{c}{\approx} \textsc{hyb}_2$: Indistinguishability of hybrids follows from reduction to the security of the underlying OT scheme [EGL85].

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.1}$: The difference between the hybrids is that the commitment to λ_w^1 for each output wire w, is created for a dummy value in $\text{HYB}_{3.1}$. The indistinguishability follows via reduction to the hiding property of eCom.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_{3.2}$: The difference between the hybrids is that in $\text{HYB}_{3.2}$, commitment to λ_w^1 for each output wire w, is created for a dummy value and later equivocated using o_w^1 computed via where $\mathsf{o}_w^1 = \mathsf{Equiv}(\mathsf{c}_w^1, \mathsf{o}_w'^1, \lambda_w^1, t)$ where t is the trapdoor for the commitment c_w^1. Indistinguishability follows via reduction to the hiding property of eCom.

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The only difference between the hybrids is that, in HYB_3, the protocol aborts if for some output wire w and index $j \in \mathcal{S}_g$, k_{w,b_w}^j of the received $\mathbf{Y}$ does not match with either $(k_{w,0}^j, k_{w,1}^j)$ or the keys $\{k_{w,b_w}^j\}_{j \in \mathcal{S}_g}$ in $\mathbf{Y}$ do not map to the same b_w whereas in HYB_4, the protocol results in abort if the received $\mathbf{Y}$ does not match the one created with simulated GC. By security of the garbling scheme, P_4 could have forged such a $\mathbf{Y}$ only with negligible probability. $\square$

Chapter 9

Empirical Results

In this chapter, we elaborate the empirical results of our protocols. We use the circuits of AES-128 and SHA-256 as benchmarks. We begin with the details of the setup environment, both hardware and software and then give a detailed comparison of efficiency.

9.1 Setup

9.1.1 Hardware Details

We provide experimental results both in LAN and WAN (high latency) settings. For the purpose of LAN, our system specifications include a 32GB RAM; an Intel Core $i7-7700-4690$ octa-core CPU with 3.6 GHz processing speed with AES-NI support from the hardware. For WAN, we have employed Microsoft Azure D4s_v3 cloud machines with instances located in West US, South India, East Australia, South UK and East Japan. The average bandwidth measured using the *iperf* testing tool corresponds to $169 Mbps$. The slowest link has a round trip time (RTT) of 277 `ms` between East Australia and South UK. RTT denotes the time required to send a packet from source to destination and subsequently an acknowledgment back from destination to source. But the transfer of a packet involves only one way communication from source to destination. So the delay that we consider is half of RTT which is 138.5 `ms` for our slowest link (present between garblers $P_3 - P_4$). The following are the maximum delays for each garbler for one way communication: P_1: 102 `ms`, P_2: 101 `ms`, P_3: 138 `ms`, P_4: 138.5 `ms`. (Garbler P_4 is not used in 4PC as only 3 garblers are present) For the evaluator, the maximum delay is close to 112 `ms`. The tables indicate the average delay for the role of garbler which turns out to be between $114 - 120$ `ms`.

9.1.2 Software Details

For efficiency, the technique of free-XOR is enabled and the implementation is carried out using *libgarble* library licensed under GNU GPL license. This library leverages the use of AES-NI instructions provided by the underlying hardware.We additionally use openSSL 1.02g library for SHA to instantiate our commitments. The operating system used is Ubuntu 16.04 (64-bit). Our code follows the standards of C++11 and multi-threading is enabled on all cores for improved results. Communication is done using sockets whose maximum size is set to 1 MB and a connection is established between every pair of parties to emulate a complete network consisting of pair-wise private channels.

9.2 Comparison

We compare our results in the high-latency network with the relevant ones. We highlight the following parameters for analysis: computation time (**CT**)– the time spent computing across all cores, runtime (CT + network time) in terms of **LAN**, **WAN** and communication (**CC**). The network time emphasizes the influence of rounds and communication size taking into account the proximity of servers. The state of the art in 3PC [MRZ15, BJPR18] and 4PC [BJPR18] with honest majority achieving various notions of security, incur significantly less overhead compared to our setting since they tolerate one corruption which aids in usage of inexpensive Yao's garbled circuits [BHR12] and fewer rounds. Thus, the closest result to our setting is [CGMV17] in terms of both number of corruptions and tools used. Below we make a detailed comparison with it.

9.2.1 Analysis of 5PC

For fair analysis, we instantiate the protocol of [CGMV17] in our environment and use the semi-honest 4DG scheme (Fig 3.5) in place of [BLO16] that they rely on. However, we also instantiate [CGMV17] with the 4DG scheme of [BLO16] to emphasize the saving in computation time that occurs with the use of Garble_4 in place of the scheme of [BLO16]. The tables highlight average values distinctly for the role of a garbler ($P_g, g \in [4]$) and the evaluator (P_5). The results for [CGMV17], ua5PC, fair5PC appear in Table 9.1. Table 9.2 depicts the results for god5PC. While having the round complexity of 8 and achieving stronger security, ua5PC and fair5PC incur an overhead of at most 0.2 MB overall for both circuits over [CGMV17]. The overhead in both protocols is a result of the proof of origin of output super-key $\mathbf{Y}$ and exchange of $\mathbf{Y}$ among garblers. Additionally, in fair5PC, the *commit-then-open* trick on output mask bits constitutes extra communication. For the necessary robust broadcast channel in god5PC, we use

Dolev Strong [DS83] (DS) to implement authenticated broadcast and fast elliptic-curve based schemes [BDL+12] to realize public-key signatures therein. These signatures have a one-time setup to establish public-key, private-key for each party. We do the same for robust 3PC of [BJPR18] for empirical purposes.

When instantiated with DS broadcast, the round complexity for honest run of GOD is 12 (in the presence of 4 broadcasts) and the shown WAN overhead in Table 9.2 over [CGMV17] captures this inflation in rounds. For the sake of implementation of all protocols (including [CGMV17] for fair comparison), we have adopted parallelization wherever possible. Next, if we observe god5PC, Table 9.2 indicates that the pairwise communication (CC) of god5PC protocol is almost on par with that of [CGMV17] in Table 9.1 (and less than fair5PC). This is because, the honest run of our god5PC is almost same as [CGMV17] except for the input commit routine and the use of broadcast. The input commit routine can be parallelized with the process of garbling to minimize number of interactions. This implies that the majority overhead is mainly due to the use of broadcast. The implementation of DS broadcast protocol is done by first setting up public-key, private key pair for each party involved. Each message sent by the broadcast sender is then agreed upon by the parties by running 3 (t+1) rounds. If multiple independent broadcasts exist in one round, they are run parallelly. Also, any private communication that can be sent along with the broadcast data is also parallelized for improved round complexity.

The broadcast communication is kept minimal and independent of the circuit, input and output size. As a result, the total data to be broadcasted constitutes only 1.73 KB of the total communication. In the honest run, when the adversary does not strike, the overall overhead amounts to a value of at most 1.2 s in WAN over [CGMV17]. The worst case run in god5PC occurs when the adversary behaves honestly throughout but only strikes in the final broadcast of $\mathbf{Y}$ and a 3PC instance is run from that point. In this case, the overall WAN overhead is at most 2.5 s over [CGMV17]. This overhead is justified considering the strength of security that the protocol offers when compared to [CGMV17]. Also, the overheads in LAN and communication are quite reasonable.

In the fair5PC, the higher overhead of 0.2 MB than honest run of god5PC is the result of commitments on output wire masks and circulation of $\mathbf{Y}$ and proof of origin of $\mathbf{Y}$ in the output phase as explained above. Also, fair5PC protocol involves 3 sequential rounds for output phase compared to single communication of $\mathbf{Y}$ by P5 in [CGMV17] and in god5PC. Note that in the LAN setting, the RTT is of the order of microseconds for one packet send. Our observations show that, in the LAN setting, the RTT sensitively scales with the communication size whereas in WAN, the RTT hardly varies for small increase in communication. For instance, we have noted that, in LAN, the average RTT for 1 KB, 8 KB, 20 KB, 80 KB is 280μs, 391μs, 832μs, 1400μs respectively, whereas in WAN the RTT for these communication sizes does not vary. This implies that two transfers of 1 KB data consumes less time than a single transfer of 20 KB data in LAN. All the above reasons collectively justify the slight difference in the LAN time. Having said that, we believe that WAN being a better comparison measure in terms of both communication data and round complexity, aptly depicts the overhead of all our protocols over [CGMV17].

Table 9.1: Computation time (**CT**), LAN run-time (**LAN**), WAN run-time (**WAN**) and Communication (**CC**) for [CGMV17], ua5PC and fair5PC for $g \in [4]$.

	Protocol	CT(ms)		LAN(ms)		WAN(s)		CC(MB)	
		P_g	P_5	P_g	P_5	P_g	P_5	P_g	P_5
AES-128	[CGMV17] (with Garble_4)	20.84	13.45	25.01	21.45	2.54	0.99	7.38	0.031
	[CGMV17] (with [BLO16])	24.4	14.17	28.56	22.17	2.58	1.0	7.38	0.03
	ua5PC	21.72	13.65	25.66	21.85	2.74	0.99	7.42	0.039
	fair5PC	21.79	13.74	26.06	22.3	2.82	1.10	7.43	0.039
SHA-256	[CGMV17] (with Garble_4)	247.69	88.23	290.38	236.53	3.44	4.78	97.26	0.062
	[CGMV17](with [BLO16])	259.99	103.54	302.6	254.21	3.58	4.8	97.26	0.06
	ua5PC	247.89	88.75	293.25	241.51	3.69	4.79	97.28	0.078
	fair5PC	249.35	88.78	301.33	242.66	3.78	4.81	97.29	0.078

Table 9.2: Computation time (**CT**), LAN run-time (**LAN**) and Communication (**CC**) and Broadcast (**BC**) for protocol god5PC for $g \in [4]$. $P_{g'}$ is the garbler and P_γ is the evaluator for worst case 3PC run.

Circuit	CT(ms)		LAN(ms)		WAN(s)		CC(MB)		BC(KB)	
	$P_g(P_{g'})$	$P_5(P_\gamma)$	$P_g(P_{g'})$	$P_5(P_\gamma)$	$P_g(P_{g'})$	$P_5(P_\gamma)$	$P_g(P_{g'})$	$P_5(P_\gamma)$	$P_g(P_{g'})$	$P_5(P_\gamma)$
AES-128	21.93 (+1.12)	13.34 (+0.91)	28.95 (+2.39)	24.19 (+2.1)	3.70 (+1.02)	1.76 (+1.1)	7.41 (+0.15)	0.032 (+0.002)	10.416 (+4.03)	10.064 (+4.06)
SHA-256	249.91 (+11.63)	90.83 (+9.76)	295.3 (+14.5)	241.83 (+11.9)	4.5 (+1.42)	5.6 (+1.51)	97.27 (+3.074)	0.064 (+0.004)	10.416 (+4.03)	10.064 (+4.06)

9.2.2 Analysis of 4PC

As efficiency studies considering mixed adversary is limited and no relevant literature exists for small party domain to the best of our knowledge, we mainly compare with MPC with small population in the traditional honest majority. In the mixed model protocols, the closest work to ours is that of [CGMV17] which explores selective abort with 5 parties against 2 active corruptions since we rely on the tools of SD, AOT, distributed garbling similar to theirs. In the 4-party domain, the state of the art protocol of [BJPR18] achieves GOD with 1 corruption.

Since, the corruption scenario of our mixed protocols lies between the above two results, we show a detailed comparison with them.

Table 9.3 provides the comparison of fair4PC and god4PC with the 4PC GOD of [BJPR18] and selective abort protocol of [CGMV17]. We implement the protocols of [BJPR18, CGMV17] in our environment for fair comparison. From the table, observe that, the performance of our protocol lies between that of [BJPR18] with one active corruption and [CGMV17] with 2 active corruptions (as expected). The overhead over [BJPR18] comes from distributed garbled circuit used in our mixed protocols (due to 2 corruptions) as compared to the use of inexpensive Yao's garbled circuit (due to only 1 corruption), thereby minimizing the communication and rounds. We save over [CGMV17] due to the difference in the number of parties. Nevertheless, our protocols achieve stronger security of fairness and GOD while going beyond strict honest majority as opposed to the weakest security of selective abort achieved by [CGMV17] in honest majority, thus proving ours are better suited to practical systems than [CGMV17]. Also, the efficiency gap between [BJPR18] and [CGMV17] reflects the difficulty in moving from single to 2 corruption in the honest majority setting and the same is carried over to the dishonest majority setting of ours.

Table 9.3: Computation time (**CT**), LAN run-time (**LAN**), WAN run-time (**WAN**) and Communication (**CC**) for [CGMV17], fair4PC and god4PC protocol where $g \in [3]$ and P_e denotes the evaluator.

	Protocol	**CT(ms)**		**LAN(ms)**		**WAN(s)**		**CC(MB)**	
		P_g	P_e	P_g	P_e	P_g	P_e	P_g	P_e
AES-128	[BJPR18]	1.44	0.87	1.95	1.48	0.84	0.87	0.16	0.007
	[CGMV17] (with Garble_4)	20.84	13.45	25.01	21.45	2.54	0.99	7.38	0.031
	fair4PC	16.91	12.68	22.08	20.88	2.17	0.99	5.56	0.039
	god4PC	17.3	12.76	22.47	20.94	2.53	1.10	5.58	0.039
		(+1.05)	(+0.84)	(+1.4)	(+1.04)	(+0.24)	(+0.15)	(+0.3)	(+0.002)
SHA-256	[BJPR18]	13.97	10.81	17.68	16.72	1.23	1.28	3.02	0.014
	[CGMV17] (with Garble_4)	247.69	88.23	290.38	236.53	3.44	4.78	97.26	0.062
	fair4PC	209.69	65.27	267.24	189.24	2.94	3.79	85.58	0.02
	god4PC	210.53	68.82	273	190.82	3.40	4.24	85.62	0.02
		(+13.5)	(+9.5)	(+15.48)	(+10.8)	(+0.25)	(+0.16)	(+3)	(+0.004)

Table 9.4 provides a unified view of the overall maximum latency in terms of each parameter and total communication of all protocols implemented with Garble in Chapter 3. The bracketed values indicate the additional overhead involved in the worst case run of god5PC.

Note that the overhead for SHA-256 is higher compared to AES-128 for 5PC. This difference maps to the circuit dependent communication involving the inputs and output. Since SHA is a huge circuit compared to AES, the increase is justified. However, the percentage overheads get better for SHA compared to AES. Besides, the factor of additional communication over-

Table 9.4: The total computation time (**Total CT**), maximum latency in LAN run-time (**LAN**) and WAN run-time (**WAN**) and total communication (**Total CC**) of all parties for [CGMV17] and our protocols using $\mathsf{Garble}_3/\mathsf{Garble}_4$. The figures in brackets indicate the increase for the worst case run of god5PC and god4PC.

	LAN(ms)						WAN(s)						Total CC(MB)					
Circuit	[CGMV17]	ua5PC	fair5PC	god5PC	fair4PC	god4PC	[CGMV17]	ua5PC	fair5PC	god5PC	fair4PC	god4PC	[CGMV17]	ua5PC	fair5PC	god5PC	fair4PC	god4PC
AES-128	25.01	25.66	26.06	28.95 (+ 2.39)	22.08	22.47 (+ 1.4)	2.54	2.74	2.82	3.7 (+ 1.1)	2.17	2.53 (+ 0.24)	29.55	29.71	29.75	29.72 (+ 0.32)	16.72	16.78 (+ 0.3)
SHA-256	290.38	293.25	301.33	295.3 (+ 14.5)	267.24	273 (+ 15.48)	4.78	4.79	4.81	5.6 (+ 1.51)	3.79	4.24 (+ 0.25)	389.12	389.2	389.24	389.19 (+ 6.15)	256.76	256.88 (+ 3.0)

head incurred by our protocols for SHA when compared to AES is far less than the factor of increase in the total communication for SHA over AES in [CGMV17] thus implying that the performance of our protocols improves with larger circuits. Further, based on our observation and in [CGMV17], using AOT instead of OT extension eliminates the expensive public key operations needed even for the seed OTs between *every pair* of garblers. Further, AOT needs just 1 round whereas OT extension needs more. All these factors lead to the improvement of our $\mathsf{Garble}_3, \mathsf{Garble}_4$ over [WRK17] which relies on large number of Tiny OTs [NNOB12] to perform authentication.

Chapter 10

Summary of the thesis and Future Scope

10.1 Summary of the Thesis

The thesis began with the introduction to the area of Secure Multi-party Computation, the threat models and the literature most relevant to our work. Then we presented the security model and the primitives used. Next we presented the efficient building blocks and distributed garbling for our five-party and for-party protocols. After presenting some preliminaries and building blocks, we described our main results. Prior to presenting our results in detail, we revisited the state-of-the-art protocol on which all our protocols are inspired from. All the formal constructions were followed by a rigorous security proof. Finally we discussed the empirical results of our protocols compared to the state-of-the-art and their suitability to practical systems. Specifically,

- Our protocols, ua5PC and fair5PC incur an overhead of at most 0.2 MB overall for both circuits over [CGMV17]. Despite using broadcast, our god5PC protocol incurs an overall WAN overhead of at most 2.5 s over [CGMV17]. Our empirical findings emphasize that the stronger security notions can be achieved with practical efficiency at an expense that is not too far from the result of [CGMV17] achieving least desired security of selective abort.

- Our protocols, fair4PC and god4PC incur overhead over [BJPR18] due to the use of distributed garbled circuit in our mixed protocols (due to 2 corruptions) as compared to the use of inexpensive Yao's garbled circuit (due to only 1 corruption), thereby minimizing the communication and rounds. However, we save over [CGMV17] due to the difference in the number of parties. Nevertheless, our protocols achieve stronger security of fairness and GOD while going beyond strict honest majority as opposed to the weakest security of selective abort achieved by [CGMV17] in honest majority, thus proving our threat models are better suited to practical systems.

10.2 Future Scope

The paramount importance of stronger security notions in practical systems makes the efficiency study of security notions interesting. The following list mentions a few possible directions for future work.

- Minimizing the round complexity while preserving / improving the efficiency of our five-party protocols.
- Efficient construction of four-party protocols achieving stronger security notions that satisfy all corruption cases in the condition $2t_a + t_p < n$ that is a single protocol that achieves stronger security notions while tolerating any of the following corruption cases: simultaneous 1 active and 1 passive corruption or 3 passive corruptions or 1 active corruption.

Bibliography

[ABF+17] 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 *IEEE Symposium on Security and Privacy*, pages 843–862, 2017. 3, 4

[ABT19] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Degree 2 is complete for the round-complexity of malicious MPC. pages 504–531, 2019. 2

[ACGJ18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *CRYPTO*, pages 395–424, 2018. 2

[ACGJ19] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In *EUROCRYPT*, 2019. 2

[ACJ17] Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO*, pages 468–499, 2017. 53

[ADMM14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *IEEE Symposium on Security and Privacy*, pages 443–458, 2014. 2

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *SIGSAC*, pages 805–817, 2016. 1, 3, 4

[AJL+12] 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 *EUROCRYPT*, pages 483–501, 2012. 1

[BCD+09] 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 *FC*, pages 325–343, 2009. 2

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, pages 77–89, 2012. 119

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011. 1

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *CRYPTO*, pages 663–680, 2012. 1, 2

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988. 1, 2

[BH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In *ASIACRYPT*, pages 376–392, 2007. 1

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, pages 213–230, 2008. 1

[BHKL18] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants. CCS '18, pages 695–712, 2018. 1

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796, 2012. 11, 58, 65, 118

[BIK+17] 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 *ACM CCS*, 2017. 2

[BJMS18] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: laziness leads to GOD. *IACR Cryptology ePrint Archive*, 2018:580, 2018. 2, 53

[BJPR18] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. Fast secure computation for small population over the internet. CCS '18, pages 677–694, 2018. 1, 3, 4, 6, 36, 55, 58, 60, 63, 64, 65, 67, 68, 69, 70, 72, 76, 118, 119, 120, 121, 123

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, pages 421–439, 2014. 2

[Bla06] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew Robshaw, editor, *Fast Software Encryption*, pages 328–340, 2006. 13

[BLO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *CCS*, pages 578–590, 2016. 4, 12, 22, 25, 27, 31, 34, 118, 120

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008. 3

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990. 1, 12

[BNDDS87] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, 1987. 78, 81

[BTW12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In *FC*, pages 57–64, 2012. 2, 3

[CCG+19] 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. https://eprint.iacr.org/2019/216. 53

[CCPS19] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh. ASTRA: High-throughput 3PC over Rings with Application to Secure Prediction. In *IACR Cryptology ePrint Archive*, 2019. 4

[CDG87] David Chaum, Ivan Damgård, and Jeroen Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, pages 87–119, 1987. 1

[CDI05] R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC*, pages 342–362, 2005. 16

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO*, pages 34–64, 2018. 4

[CGJ+17] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *CCS*, pages 719–728, 2017. 2

[CGMV17] Nishanth Chandran, Juan A. Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, constant-round and actively secure MPC: beyond the three-party case. In *CCS*, pages 277–294, 2017. vii, xii, 1, 2, 3, 4, 5, 6, 7, 13, 18, 19, 30, 34, 35, 36, 39, 40, 41, 48, 50, 53, 78, 80, 84, 118, 119, 120, 121, 122, 123, 124

[Cha89] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO*, 1989. 2

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, pages 141–150, 1998. 14

[CKMZ14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *CRYPTO*, pages 513–530, 2014. 4

[CL14] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT*, pages 466–485, 2014. 5, 7, 9, 10, 55

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986. 1, 2, 3

[DDWY93] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 1993. 2

[DGK09] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to 'efficient and secure comparison for on-line auctions'. *IJACT*, 2009. 2

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *CRYPTO*, pages 378–394, 2005. 2

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006. 2

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007. 1

[DO10] Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, pages 558–576, 2010. 1

[DOS18] Ivan Damgård, Claudio Orlandi, and Mark Simkin. Yet another compiler for active security or: Efficient MPC over arbitrary rings. In *CRYPTO*, pages 799–829, 2018. 2, 4

[DPSZ12a] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO*, pages 643–662, 2012. 1

[DPSZ12b] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012. 1

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 1983. 5, 119

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 1985. 6, 17, 25, 78, 80, 91, 95, 99, 103, 109, 112, 115

[EOP+19] H. Eerikson, C. Orlandi, P. Pullonen, J. Puura, and M. Simkin. Use your brain! arithmetic 3pc for any modulus with active security. *IACR Cryptology ePrint Archive*, 2019. 4

[FHM98] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In *CRYPTO*, 1998. 2

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *EUROCRYPT*, pages 225–255, 2017. 1, 4

[Gei07] Martin Geisler. Viff: Virtual ideal functionality framework. `http://viff.dk`, 2007. 3

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014. 53

[GIKR02] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *CRYPTO*, 2002. 2

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, pages 63–82, 2015. 2, 10, 53

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987. 1

[GRW18] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. Secure computation with low communication from cross-checking. *IACR Cryptology ePrint Archive*, 2018:216, 2018. 4

[HHPV18] Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In *CRYPTO*, 2018. `https://eprint.iacr.org/2017/1056`. 53

[HKT11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In *STOC*, pages 89–98, 2011. 13

[HLM13] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In *CRYPTO*, 2013. 3

[HMZ08] Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE : Unconditional and computational security. In *ASIACRYPT*, 2008. 2

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO*, pages 359–378, 2015. 2, 3, 4

[IKP+16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *CRYPTO*, 2016. 53

[ISN89] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 1989. 16

[KMO01] Jonathan Katz, Steven Myers, and Rafail Ostrovsky. Cryptographic counters and applications to electronic voting. In *EUROCRYPT*, 2001. 2

[LADM14] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-scale secure multiparty computation. In *Programming Languages and Systems*, pages 8–26, 2014. 3

[LP04] Yehuda Lindell and Benny Pinkas. A proof of yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. https://eprint.iacr.org/2004/175. 65, 86, 90, 91, 94, 95, 98, 99

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, pages 319–338, 2015. 1

[MR18] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. *IACR Cryptology ePrint Archive*, 2018:403, 2018. 2, 3

[MRSV17] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. PICS: private image classification with SVM. *IACR Cryptology ePrint Archive*, 2017:1190, 2017. 3

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *CCS*, pages 591–602, 2015. 1, 3, 4, 65, 80, 118

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991. 14

[NBK15] Divya G. Nair, V. P. Binu, and G. Santhosh Kumar. An improved e-voting scheme using secret sharing based secure multi-party computation. *CoRR*, 2015. 2

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, 2012. 1, 122

[NV18] Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority MPC by batchwise multiplication verification. In *ACNS*, pages 321–339, 2018. 4

[PR18] Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In *CRYPTO*, pages 425–458, 2018. 1, 2, 3, 4

[PRS02] Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *(FOCS*, 2002. 16

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In *EUROCRYPT*, pages 260–289, 2017. 2

[PW09] Rafael Pass and Hoeteck Wee. Black-box constructions of two-party protocols from one-way functions. In *TCC*, 2009. 15, 16

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989. 1, 2

[Ros04] Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In *TCC*, 2004. 16

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE*, pages 371–388, 2004. 17

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949. 13

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *CCS*, pages 39–56, 2017. 1, 4, 18, 122

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982. 1, 3, 6, 65, 78, 79, 82, 85, 86, 90, 91, 95, 99

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, pages 220–250, 2015. 6