Fast Actively Secure Five-Party Computation with Security Beyond Abort*

Megha Byali[1], Carmit Hazay † [2], Arpita Patra ‡ [3], and Swati Singla[4]

[1]Indian Institute of Science. Email: megha@iisc.ac.in
[2]Bar-Ilan University. Email: carmit.hazay@biu.ac.il
[3]Indian Institute of Science. Email: arpita@iisc.ac.in
[4]Indian Institute of Science. Email: swatis@iisc.ac.in

Abstract

Secure Multi-party Computation (MPC) with small population and honest majority has drawn focus specifically due to customization in techniques and resulting efficiency that the constructions can offer. In this work, we investigate a wide range of security notions in the five-party setting, tolerating two active corruptions. Being constant-round, our protocols are best suited for real-time, high latency networks such as the Internet.

In a minimal setting of pairwise-private channels, we present efficient instantiations 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). With the presence of an additional broadcast channel (known to be necessary), we present a construction with *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*. 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.

*This article is the full and extended version of an earlier article to appear in ACM CCS 2019

†This author was supported by European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS), by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and by ISF grant 1316/18.

‡The author would like to acknowledge financial support by Tata Trust Travel Grant 2019 and SERB Women Excellence Award 2017 (DSTO 1706).

Contents

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. The vast literature of MPC can be broadly 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).

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 paper, we consider MPC for 5 parties (5PC) with 2 corruptions and treat it with securities of unanimous abort, fairness and GOD, at an expense that is not too far from the result of [CGMV17]

achieving least desired security of selective abort.

1.1 Related Work

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.

The work most relevant to ours 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]. 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.

1.2 Our Contribution

In the regime of low-latency protocols which is of interest to us, the 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 paper is yet another attempt in this direction, focused on the 5-party setting.

We present efficient, constant-round 5PC protocols with honest majority that achieve security notions ranging from unanimous abort to GOD, solely relying on symmetric-key primitives. 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]. Our theoretical findings are backed with implementation results with the choice of benchmark circuits AES-128 and SHA-256.

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 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.

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. (*: the total number of rounds is calculated plugging in the state of the art robust 3PC [BJPR18]. The rounds for GOD is stated assuming broadcast channel availability in ours and [BJPR18]).

Security	**Rounds**	**BC**	**LAN (ms)**	**WAN (s)**	**CC (MB)**
unanimous abort	8	✗	0.65-2.87	0.2-0.01	0.16-0.09
fairness	8	✗	1.05-10.95	0.28-0.03	0.2-0.13
GOD (honest run)	6	✓ [CL14]	3.94-4.92	1.16-0.82	0.17-0.07
GOD (worst case)	12*	✓ [CL14]	6.33-19.42	2.26-2.33	0.49-6.22

All protocols barring the one with GOD maintain the same circuit-dependent communication as [CGMV17]. The GOD protocol costs two circuit-dependent communication, one in 5PC and one in 3PC, the latter amongst a smaller instance of 3 parties. This is reflected in the cost of worst case run of our GOD protocol. For all other constructions, 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 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 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.

2 Preliminaries

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 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 our setting i.e., the corrupt parties can arbitrarily deviate from the protocol specification. 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 all our protocols is proved in the standard real/ideal world paradigm. Appendix A elaborates on the functionalities and security definitions. Below we discuss the primitives that we use.

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. The properties to be satisfied by a commitment scheme are: *correctness* (a commitment opens to the correct message), *hiding* (a corrupt receiver cannot derive any information about the message from the commitment alone) and *binding* (a corrupt committer cannot open a given commitment to multiple messages). We use instantiations based on injective one-way functions that ensure a strong binding even if the public parameter is arbitrarily chosen by adversary.

For our fair protocol, 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 the property of equivocation. The formal definitions and the instantiations appear in Appendix B.

Secret Sharing Schemes We use *additive sharing* and *replicated secret sharing* (RSS) [CDI05, ISN89]. 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 $\mathcal{T}_l$ (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 $i \notin \mathcal{T}_l$. This ensures that t parties alone of any $\mathcal{T}_l$ cannot retrieve x. We use a 4-party RSS with $t = 2$ 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 where $\mathcal{T}_l = \{P_i, P_j\}$ is missing from the view.

3 Distributed Garbling and More

At the heart of our 5PC lies a 4-party distributed garbling (4DG) and a matching evaluation protocol tolerating arbitrary *semi-honest* corruptions. Garbling is done distributively amongst the garblers $\{P_1, P_2, P_3, P_4\}$ and P_5 enacts the sole evaluator. Our 4DG 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 seed distribution technique.

3.1 Seed Distribution (SD)

In the 4DG, 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 $\mathcal{F}_{\mathsf{SD}}$ is depicted in Fig 1 and is realized differently in each of our protocols based on the required security guarantee (fairness 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 $\mathsf{s}_1, \mathsf{s}_3, \mathsf{s}_4$.

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 $\mathcal{C}$ be the set of corrupt parties. Corrupted parties $P_j \in \mathcal{C}$ may send the trusted party $(\mathsf{Input}, \mathsf{s}_j/\perp)$ as instructed by the adversary.
On message $(\mathsf{Input}, *)$ from garbler $P_g, g \in [4] \setminus \mathcal{C}$ and $(\mathsf{Input}, \{\mathsf{s}_j/\perp\}_{j\in\mathcal{C}})$ from adversary, sample s_i on behalf of every honest P_i and send s_g (or $\perp$ as given by adversary) to each party in $\mathcal{S}_g$.

Figure 1: Functionality $\mathcal{F}_{\mathsf{SD}}$

3.2 Attested Oblivious Transfer (AOT)

The AOT protocol [CGMV17] can be viewed as an OT between a sender and a receiver with an 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. AOT functionality is recalled in Fig 2.

P_s acts as sender, P_r acts as receiver and P_{a_1}, P_{a_2} act as attesters.
- On input message (Sen, m_0, m_1) from P_s, record (m_0, m_1) and send (Sen, m_0, m_1) to P_{a_1} and P_{a_2} and Sen to the adversary.
- On input message (Rec, b) from P_r, where $b \in \{0,1\}$, record b and send (Rec, b) to P_{a_1} and P_{a_2} and Rec to the adversary.
- On input message (Att, m_0^j, m_1^j, b^j) from $P_{a_j}, j \in [2]$, if (Sen, sid, $*$, $*$) and (Rec, $*$) have not been recorded, ignore this message; otherwise, record ($m_0^{a_j}$, $m_1^{a_j}$, b^{a_j}) and send Att to the adversary.
- On input message 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 (Output, $\perp$) to P_r; else send (Output, m_b) to P_r.
- On input message abort from the adversary, send (Output, $\perp$) to P_r.

Figure 2: Functionality $\mathcal{F}_{\mathsf{4AOT}}(P_s, P_r, \{P_{a_1}, P_{a_2}\})$

3.3 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. 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_{u,b_u}^g, k_{v,b_v}^g 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 $\{k_{w,0}^g \oplus \Delta^g(x_u x_v \oplus \lambda_w)\}_{g\in[4]}$ that corresponds to the blinded (masked) bit $x_u x_v \oplus \lambda_w$ on wire w.

Definition 3.2. 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 $\{k_{w,0}^g \oplus \Delta^g((b_u \oplus \lambda_u)(b_v \oplus \lambda_v) \oplus \lambda_w)\}_{g\in[4]}$. 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 3.3. 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, $\{c_\gamma^g\}_{g\in[4]}$, where P_g contributes c_γ^g that encrypts its additive share of each key in $\{k_{w,0}^g \oplus \Delta^g((b_u \oplus \lambda_u)(b_v \oplus \lambda_v) \oplus \lambda_w)\}_{g\in[4]}$.

To compute the additive sharing of super-key $\{k_{w,0}^g \oplus \Delta^g((b_u \oplus \lambda_u)(b_v \oplus \lambda_v) \oplus \lambda_w)\}_{g\in[4]}$ 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) $k_{w,0}^g \oplus \Delta^g\lambda_\gamma$ 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). Partitioning the set of all super-CTs into its 4 constituent CTs, we can view the GC as $GC^1 \,||\, GC^2 \,||\, GC^3 \,||\, GC^4$ where gth partition is contributed by garbler P_g.

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 gate evaluated using free-XOR. For an AND gate with input wires u,v, P_5, with input super-keys $\{(k_{u,b_u}^g, k_{v,b_v}^g)\}_{g\in[4]}$ 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 $x_ux_v \oplus \lambda_w$ and the blinded output bit itself. The blinded bits for output wires give clear output when XORed with their respective masks.

3.4 Distributed Garbling 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:*

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 $\Delta^{g'}\lambda_\gamma$, $g' \in [4]$.

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

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

Figure 3: Functionality $\mathcal{F}_{\mathsf{GC}}$

The standard OTs used to compute each cross-term $\lambda_u^g\lambda_v^{g'}$, $\Delta^g\lambda_\gamma^{g'}$ $(g \neq g')$ 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 (Appendix **??**). 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_{uv}^1 as indicated in the table.

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

Our final garbling and evaluation protocols appear in Figs 4-5. The correctness proof appears in Lemma 3.4.

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

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_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 $\lambda_{uv} = \lambda_u\lambda_v = (\oplus_{g\in[4]}\lambda_u^g)(\oplus_{g\in[4]}\lambda_v^g)$. 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 $k \in \mathcal{S}_g$, 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 $P_s, s = [4]\backslash\{g, j, k\}$ 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) 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 4: Protocol Garble()

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 compute $b_w = \oplus_{g\in[4]}\lambda_\gamma^g$ and $k_{w,b_w}^g = k_w^g \oplus (\oplus_{g'\neq g}[\Delta^g\lambda_\gamma]_{g'})$ after decrypting every CT c_γ^g in the γth super-CT as follows:
  $(\lambda_\gamma^g||\{[\Delta^{g'}\lambda_\gamma]_g\}_{g'\neq g}||k_w^g) := \mathsf{F}_{k_{u,b_u}^g, k_{v,b_v}^g}(j||g) \oplus c_\gamma^g$.

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

Figure 5: Protocol Eval()

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.

3.5 Attested OT Instantiation

For the attested OT functionality $\mathcal{F}_{\mathsf{4AOT}}$ defined in Fig 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 message received from P_s, P_r amongst themselves for 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.

When coupled with seed distribution, the standalone realization of $\mathcal{F}_{\mathsf{4AOT}}$ can be simplified as follows: The attesters are chosen s.t they possess the inputs (derived from seed) of both sender and receiver. For instance, when $P_s = P_1, P_r = P_2$, the attesters are P_3, P_4 and the inputs of the 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 commitment c_b. All these steps can be done 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 6.

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 $(\mathsf{Com}, \mathsf{Open})$ (Appendix B).

– 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 6: Protocol $\Pi_{\mathsf{4AOT}}(P_s, P_r, \{P_{a_1}, P_{a_2}\})$ for Garble

The protocol realization specific to god5PC is presented in Fig 7. 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.

3.6 Correctness and Security of 4DG

Lemma 3.4. *The protocols* Garble *and* Eval *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

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 $(\mathsf{Com}, \mathsf{Open})$ (Appendix B).

– 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 7: Protocol $\Pi_{\mathsf{4AOTGOD}}(P_s, P_r, \{P_{a_1}, P_{a_2}\}, P_a)$

$\mathsf{g}(\lambda_u, \lambda_v) = \mathsf{g}(x_u, x_v)$ 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:

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

The evaluator computes $b_w = \oplus_{g\in[4]}\lambda^g_1 = \mathsf{g}(x_u, x_v)$ 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.

□

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.

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 $(\lambda^j_w, j \notin \mathcal{S}_g)$ 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^{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^{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.

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 $\mathsf{H}(\mathsf{proof})$ 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.

4.1 The construction

We present the formal protocol in Fig 8. The garblers perform a one-time SD as in [CGMV17], which can be used for multiple runs. Circuit garbling is done as in Fig 4. 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.

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

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 NICOM $(\mathsf{Com}, \mathsf{Open})$, an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$, Garble (Fig 4), Eval (Fig 5), Collision Resistant Hash H (Appendix B).
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 the following commitments for every circuit output wire w using randomness from $\mathsf{s}_j, j \in \mathcal{S}_g$: $\{(\mathsf{c}^j_w, \mathsf{o}^j_w) \leftarrow \mathsf{eCom}(\mathsf{epp}^j, \lambda^j_w)\}_{j\in\mathcal{S}_g}$. P_g sends $\{(\mathsf{epp}^j, \mathsf{c}^j_w)\}_{j\in\mathcal{S}_g}$ to all.
- $P_i \in \mathcal{P}$ aborts if it receives mismatched copies of $(\mathsf{epp}^j, \mathsf{c}^j_w), j \in [4]$ 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 $\lambda^j_w, j \in \mathcal{S}_l$ to P_g. P_g aborts if it receives mismatched copies for some λ^j_w. Else, P_g computes $\lambda_w = \oplus_{j\in[4]} \lambda^j_w$ and $b_w = x_w \oplus \lambda_w$. P_g sends $(b_w, \{k^j_{w,b_w}\}_{j\in\mathcal{S}_g})$ to P_5. To send $k^j_{w,b_w}, j \in [4] \setminus \mathcal{S}_g$ (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 $K_l = k^j_{w,b^l_w} \oplus \beta_l \oplus \delta_l$ to P_5 who sets $k^j_{w,b_w} := \oplus_l K_l$.
- For input wire w corresponding to P_5's input shares, let $\{k^g_{w,0}, k^g_{w,1}\}_{g\in[4]}$ be the keys derived from seeds $\{\mathsf{s}_g\}_{g\in[4]}$. Each $P_g, g \in [4]$ computes commitments on these as: for $b \in \{0,1\}, j \in \mathcal{S}_g$, $(c^j_{w,b}, o^j_{w,b}) \leftarrow \mathsf{Com}(\mathsf{pp}^j, k^j_{w,b})$ using pp^j and randomness derived from s_j and sends $\{\mathsf{pp}^j, c^j_{w,b}\}$ to P_5. P_g also sends o^j_{w,b_w} 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, $\{k^g_{w,b_w}\}_{g\in[4]}$. Let $\mathbf{X}$ to be the set of super-keys obtained.
- $\mathsf{Garble}(C)$ 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 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^j_{w,b_w} of $\mathbf{Y}$ for some output wire w does not match with either $(k^j_{w,0}, k^j_{w,1})$ 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 8: Protocol fair5PC

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 Appendix C.

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.

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

n-party Extension 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 8). The technical overview and the formal protocol appear in Fig 9 (Section 4.3).

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 (Appendix B) 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, 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.

4.2 Properties

Lemma 4.1. *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 4, 5). □

Theorem 4.2. *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. □

Theorem 4.3. *Assuming one-way permutations, the protocol of* fair5PC *securely realizes* $\mathcal{F}_{\mathsf{fair}}$ *(Fig. 17) in the standard model against a malicious adversary that corrupts at most two parties.*

The formal security proof appears in Appendix. C.

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.

4.3 n-party Extension of fair5PC

n-party Extension We first recall the conditions involved in seed distribution for n-parties elaborated in [CGMV17] to better understand the extension tolerating $t = \sqrt{n}$ 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 9.

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 9: Output Phase for n-party fairness

5 5PC with Unanimous Abort

5.1 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 (Fig 10), proof of correctness are presented below. Similar to our fair protocol, this protocol can also be extended for arbitrary n parties by modifying the output phase of uAbort5PC (Fig 10) as in Fig 11 (Section 5.3).

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

Primitives: A secure NICOM $(\mathsf{Com}, \mathsf{Open})$ (Section 2), Garble_4 (Figs. 4), Eval_4 (Fig. 5).
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^j_{w,b_w} of $\mathbf{Y}$ for some output wire w 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 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 10: Protocol uAbort5PC

Optimizations. The efficiency of uAbort5PC protocol can be boosted similar to fair5PC in both the garbling phase and communication of GC.

5.2 Properties

Lemma 5.1. *The* uAbort5PC *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 4, 5). □

Theorem 5.2. *Our* uAbort5PC *protocol runs in at most 8 rounds.*

Proof. The proof follows from the proof of Theorem 4.2. □

Theorem 5.3. *Assuming one-way permutations, our protocol* uAbort5PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{uAbort}}$ *(Fig. 18) in the standard model against a malicious adversary that corrupts at most two parties.*

The security proof is provided in Appendix D.

5.3 n-party Extension of uAbort5PC

To achieve unanimous abort for the case of n parties, all steps of the protocol uAbort5PC remain the same except the **output phase**. The seed-distribution is done as explained in Section 4.3. 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 11.

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 11: n-party Extension: Output Phase for n-party unanimous abort

6 5PC with Guaranteed Output Delivery (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. 12 and 13 respectively.

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 NICOM $(\mathsf{Com}, \mathsf{Open})$ (Appendix B), 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 $P_l, l \notin \mathcal{T}_j$.

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 12: Protocol $\mathsf{inputGOD}_i$

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 13: 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, 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 25 (Appendix F).

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 (Fig. 7). Our 3PC appears in Fig. 14. The main protocol appears in Fig. 15.

Optimizations To improve efficiency, the garbling process is optimized similar to 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

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 $i \in [5], l \in [6], k \notin \mathcal{T}_l$.
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 24 [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 $x^{ij} \oplus r^{ij}$ 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 14: Protocol god3PC

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.

6.2 Properties

Lemma 6.1. *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

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 NICOM $(\mathsf{Com}, \mathsf{Open})$, $\mathsf{inputGOD}_i$ (Fig. 12), $\mathsf{seedGOD}_g$ (Fig. 13), Garble (Fig. 4), Eval (Fig. 5) and Π_{4AOTGOD} (Fig. 7).
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. $\mathsf{Garble}(C)$ is run where Π_{AOTGOD} (Fig 7) is used instead of $\mathcal{F}_{\mathsf{4AOT}}$. Each $P_g, g \in [4]$ broadcasts $\{GC^j\}_{j\in\mathcal{S}_g}$. Each party runs 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 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 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 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 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, $\{k_{w,b_w}^g\}_{g\in[4]}$ 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 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 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 $y_w = (y_w \oplus \lambda_w) \oplus_{g\in[4]} \lambda_w^g$ and thus y. Finally, P_5 outputs y. P_5 broadcasts $\mathbf{Y}$.
- Every party P_g runs god3PC with $\mathcal{P}^3$ with $\mathcal{P}^3 = \mathcal{P} \setminus \{P_1, P_5\}$ if k_{w,b_w}^j of $\mathbf{Y}$ for some output wire w and index $j \in \mathcal{S}_g$ 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. 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 $y_w = (y_w \oplus \lambda_w) \oplus_{g\in[4]} \lambda_w^g$. Finally, P_g outputs y.

Figure 15: Protocol god5PC

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 6.2. *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 6.1). To

ensure the 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. *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 inputGOD. The honest parties are established to have committed to their inputs by the end of round 1 in inputGOD. An honest P_α obtains the output either by decoding the output super-key $\mathbf{Y}$ or via the output of god3PC (as a participant in god3PC or recipient from the 3PC committee). In the latter case, correctness follows from Lemma 6.2 and correctness of 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 inputGOD. The correctness of evaluation follows from the correctness of the garbling scheme. □

Theorem 6.4. *Assuming one-way permutations, protocol* god5PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{god}}$ *(Fig. 16) in the standard model against an active adversary that corrupts at most two parties.*

The security proof is presented in Appendix E. Since the inputs are defined prior to the garbling phase in 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 Appendix E, here, we provide intuition of GOD for completeness. The routine 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), inputs committed in inputGOD alone are used to obtain the output y thus concluding the intuition.

7 Empirical Results

In this section, 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.

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 $169Mbps$. The slowest link has a round trip time 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. 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.

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 network consisting of pair-wise private channels.

Table 1: Computation time (**CT**), LAN run-time (**LAN**), WAN run-time (**WAN**) and Communication (**CC**) for [CGMV17], uAbort5PC 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)	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
	uAbort5PC	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)	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
	uAbort5PC	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 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)

We compare our results in the high-latency network with the relevant ones. 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] and below we make a detailed comparison with it.

For fair analysis, we instantiate the protocol of [CGMV17] in our environment and use the semi-honest 4DG scheme (Section 3) 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 in place of the scheme of [BLO16]. 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 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], uAbort5PC, fair5PC appear in Table 1. Table 2 depicts the results for god5PC. While having the round complexity of 8 as in [CGMV17] and achieving stronger security, uAbort5PC 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] to implement authenticated broadcast and fast

Table 3: 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 Garble. The figures in brackets indicate the increase for the worst case run of god5PC.

Circuit	Total CT(ms)				LAN(ms)				WAN(s)				Total CC(MB)			
	[CGMV17]	uAbort5PC	fair5PC	god5PC	[CGMV17]	uAbort5PC	fair5PC	god5PC	[CGMV17]	uAbort5PC	fair5PC	god5PC	[CGMV17]	uAbort5PC	fair5PC	god5PC
AES-128	96.81	100.53	100.9	101.06 (+ 3.15)	25.01	25.66	26.06	28.95 (+ 2.39)	2.54	2.74	2.82	3.7 (+ 1.1)	29.55	29.71	29.75	29.72 (+ 0.32)
SHA-256	1078.99	1080.31	1086.18	1090.47 (+ 33.02)	290.38	293.25	301.33	295.3 (+ 14.5)	4.78	4.79	4.81	5.6 (+ 1.51)	389.12	389.2	389.24	389.19 (+ 6.15)

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 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 2 indicates that the pairwise communication (CC) of god5PC protocol is almost on par with that of [CGMV17] in Table 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 3 provides a unified view of the overall maximum latency in terms of each parameter and total communication of all protocols implemented with Garble. 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. 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 overhead 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 Garble over [WRK17] which relies on large number of Tiny OTs [NNOB12] to perform authentication.

References

[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.

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

[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.

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

[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.

[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.

[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.

[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.

[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.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, pages 169–188, 2011.

[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.

[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.

[BH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In *ASIACRYPT*, pages 376–392, 2007.

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC*, pages 213–230, 2008.

[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.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796, 2012.

[BJMS18] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: laziness leads to GOD. *IACR Cryptology ePrint Archive*, 2018:580, 2018.

[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.

[BK14] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*, pages 421–439, 2014.

[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.

[BLO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *CCS*, pages 578–590, 2016.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, pages 192–206, 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

[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.

[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.

[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.

[CDI05] R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC*, pages 342–362, 2005.

[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.

[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.

[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.

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC*, pages 141–150, 1998.

[CL14] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT*, pages 466–485, 2014.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudo-random generator. In *CRYPTO*, pages 378–394, 2005.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, pages 572–590, 2007.

[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.

[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.

[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.

[DPSZ12b] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 1983.

[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.

[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.

[Gei07] Martin Geisler. Viff: Virtual ideal functionality framework. `http://viff.dk`, 2007.

[GIKR02] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *CRYPTO*, 2002.

[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.

[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.

[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.

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO*, pages 359–378, 2015.

[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.

[LADM14] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-scale secure multiparty computation. In *Programming Languages and Systems*, pages 8–26, 2014.

[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.

[MR18] Payman Mohassel and Peter Rindal. ABY3: A mixed protocol framework for machine learning. *IACR Cryptology ePrint Archive*, 2018:403, 2018.

[MRSV17] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. PICS: private image classification with SVM. *IACR Cryptology ePrint Archive*, 2017:1190, 2017.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *CCS*, pages 591–602, 2015.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2):151–158, 1991.

[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.

[NV18] Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority MPC by batchwise multiplication verification. In *ACNS*, pages 321–339, 2018.

[PR18] Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In *CRYPTO*, pages 425–458, 2018.

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In *EUROCRYPT*, pages 260–289, 2017.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85, 1989.

[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.

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *CCS*, pages 39–56, 2017.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

A Functionalities and Security Model

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.

Each honest party P_i ($i \in [5]$) 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 ($i \in [5]$), 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 $y = f(x_1', x_2', x_3', x_4', x_5')$ and send $(\mathtt{Output}, y)$ to party P_i for every $i \in [5]$.

Figure 16: Ideal Functionality $\mathcal{F}_{\mathsf{god}}$

Each honest party P_i ($i \in [5]$) 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 $i \in [5]$ 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 $i \in [5]$, where $y = f(x_1', x_2', x_3', x_4', x_5')$.

Figure 17: Ideal Functionality $\mathcal{F}_{\mathsf{fair}}$

Each honest party P_i ($i \in [5]$) 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 $x_i' = \mathtt{abort}$.
Output to the adversary: If there exists $i \in [5]$ such that $x_i' = \mathtt{abort}$, send $(\mathtt{Output}, \perp)$ to all the parties. Else, send $(\mathtt{Output}, y)$ to the adversary, where $y = f(x_1', x_2', x_3', x_4', x_5')$.
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 18: Ideal Functionality $\mathcal{F}_{\mathsf{uAbort}}$

B Primtives

B.1 Non-Interactive Commitment Scheme

We use Non-Interactive Commitment Scheme (NICOM) characterized by two PPT algorithms (Com, Open) and are defined as:

- 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 commitment scheme should satisfy the following properties:

- *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 $\mathcal{A}$.

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 $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way permutation and $h : \{0,1\}^n \to \{0,1\}$ 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(Com, 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 $F : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ 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 $\perp$.

B.2 Equivocal Commitment Scheme

For the fair protocol, we use an Equivocal Non-Interactive Commitment Scheme (eNICOM) characterized by four PPT algorithms $(\mathsf{eCom}, \mathsf{eOpen}, \mathsf{eGen}, \mathsf{Equiv})$. The algorithms $\mathsf{eCom}, \mathsf{eOpen}$ are as defined in NICOM. The algorithms $\mathsf{eGen}, \mathsf{Equiv}$ are defined as:

– $\mathsf{eGen}(1^\kappa)$ 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.

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

B.3 Collision Resistant Hash

Consider a hash function family $\mathsf{H} = \mathcal{K} \times \mathcal{L} \to \mathcal{Y}$. The hash function H is said to be collision resistant [RS04] 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 $x_1, x_2 \in_R \{0,1\}^m$.

C Security Proof of fair5PC

We now outline the complete security proof of Theorem 4.3 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 19, 20.

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

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 **??**) 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 19: Simulator $\mathcal{S}^{12}_{\mathsf{fair5PC}}$ for 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 $\text{IDEAL}_{\mathcal{F}_{\mathsf{fair}},\mathcal{S}^{12}_{\mathsf{fair5PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{fair5PC},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1, P_2. The views are shown to be indistinguishable via a series of intermediate hybrids.

- HYB_0: Same as $\text{REAL}_{\mathsf{fair5PC},\mathcal{A}}$.
- HYB_1: Same as HYB_0 except that P_5 aborts if any decommitment for $\{k^g_{w,0}, k^g_{w,1}\}_{g\in[4]}$ corresponding to a committed share x^{52} opens to a value other than what was originally committed and held by P_2^*.
- HYB_2: Same as HYB_1 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_3: Same as HYB_2 except that $P_i, i \in \{3,4\}$ outputs $\perp$ if distributed GC cannot be successfully evaluated by P_5.

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

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The primary difference between the hybrids is that in HYB_0, P_5 aborts if the decommitments sent by P_2 corresponding to the share x^{52} output $\perp$ whereas in 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.

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

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

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

– HYB_0: Same as $\text{REAL}_{\mathsf{fair5PC}, \mathcal{A}}$.

– HYB_1: Same as 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.

– HYB_2: Same as HYB_1 except that the GC is created as simulated one with the knowledge of s_2.

– HYB_3: Same as HYB_2 except that,
  - $\text{HYB}_{3.1}$: When the execution results in `abort`, the commitment to λ_w^2 for each output wire w is created for a dummy value.
  - $\text{HYB}_{3.2}$: 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.

$\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{fair}}, \mathcal{S}_{\mathsf{fair5PC}}^{15}}$. 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_w^2 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^2 for each output wire w, is created for a dummy value and later equivocated 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. 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_5 could have forged such a $\mathbf{Y}$ only with negligible probability. □

D Security Proof of $\mathsf{uAbort5PC}$

Proof. We present the proof of Theorem 5.3 relative to its ideal functionality $\mathcal{F}_{\mathsf{uAbort}}$ (Figure **??**). We only outline the sketch of the proof, since it is very similar to the security proof of Theorem 4.3, explained in detail in Section C.

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, $\mathcal{S}_{\mathsf{uAbort5PC}}^{12}$ is described for the first case of corruption as follows: When P_1, P_2 are corrupt, $\mathcal{S}_{\mathsf{uAbort5PC}}^{12}$ acts on behalf of the honest parties. To begin with, $\mathcal{S}_{\mathsf{uAbort5PC}}^{12}$ 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 $\mathcal{S}_{\mathsf{uAbort5PC}}^{12}$ 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 $\mathcal{S}_{\mathsf{uAbort5PC}}^{12}$ on behalf of P_5 to P_2^*. $\mathcal{S}_{\mathsf{uAbort5PC}}^{12}$ 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. $\mathcal{S}^{12}_{\mathsf{uAbort5PC}}$ 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, $\mathcal{S}^{12}_{\mathsf{uAbort5PC}}$ then sends x_1, x_2 to $\mathcal{F}_{\mathsf{uAbort}}$ to obtain y which is the output of GC evaluation. $\mathcal{S}^{12}_{\mathsf{uAbort5PC}}$ then computes $\mathbf{Y}$ such that for all output wires w, each key in $\mathbf{Y}$ maps to $(y_w \oplus \lambda_w)$. $\mathcal{S}^{12}_{\mathsf{uAbort5PC}}$ 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, $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ as follows: To begin with, $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ receives s_1 from P_1^* on behalf of P_3, P_4. If the copies of s_1 received mismatch, then $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ 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. $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ has the freedom to choose s_2. $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ 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. $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ 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 $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ 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 $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ receives a valid pair $(\mathbf{Y}, \mathsf{proof}_5)$ from P_5^* on behalf of honest $P_i, i \in \{2,3,4\}$, then $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ 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 $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ sends continue to $\mathcal{F}_{\mathsf{uAbort}}$. Else, $\mathcal{S}^{15}_{\mathsf{uAbort5PC}}$ sends abort to the $\mathcal{F}_{\mathsf{uAbort}}$ to complete the simulation. □

E Security Proof of god5PC

In this section, we outline the complete security proof of Theorem 6.4 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 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 $\mathsf{inputGOD}()$ below in Figs 21.

$\mathcal{S}^{12}_{\mathsf{inputGOD}_1}$ **(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 21: Simulator $\mathcal{S}^{12}_{\mathsf{inputGOD}_1}$ (for input x_1) with actively corrupt P_1^*, P_2^*

The simulator steps for the main protocol are now presented in Figs 22, 23 respectively.

$\mathcal{S}^{12}_{\mathsf{god5PC}}$ **(**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 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.**??**) will be the output of honest parties.

Masked input bit and Key Transfer Phase.

– For $i \in \{3,4\}$ and $j \in \mathcal{S}_i \setminus \mathcal{S}_g$, 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 22: 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 $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}_{\mathsf{god5PC}}^{12}} \overset{c}{\approx} \text{REAL}_{\mathsf{god5PC},\mathcal{A}}$ when an adversary $\mathcal{A}$ corrupts P_1, P_2. The views are shown to be indistinguishable via a series of intermediate hybrids.

– HYB_0: Same as $\text{REAL}_{\mathsf{god5PC},\mathcal{A}}$.

– HYB_1: Same as 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 $\{k_{w,0}^g, k_{w,1}^g\}_{g\in[4]}$ 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 $\mathbf{Y} = \{k_{w,y_w\oplus\lambda_w}^g\}_{g\in[4]}$ 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 $\text{HYB}_4 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}_{\mathsf{god5PC}}^{12}}$. 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 $\{k_{w,0}^g, k_{w,1}^g\}_{g\in[4]}$ 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.

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

- HYB_0: Same as $\text{REAL}_{\mathsf{god5PC},\mathcal{A}}$.
- HYB_1: Same as 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.
- HYB_2: Same as HYB_1 except that the commitment to seed s_2 in $\mathsf{seedGOD}_2$ is replaced with the commitment on dummy value.
- HYB_3: Same as 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.
- HYB_4: Same as 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 $\lambda_w^2 = y \oplus (\oplus_{i \in [4], i \neq 2} \lambda_w^i)$.
- HYB_5: Same as 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 $\text{HYB}_5 = \text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}^{15}_{\mathsf{god5PC}}}$.
- HYB_6: Same as 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:

$\text{HYB}_0 \overset{c}{\approx} \text{HYB}_1$: The only difference between the hybrids is that, in HYB_1, when the execution does not result in P_1^*, P_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.

$\text{HYB}_1 \overset{c}{\approx} \text{HYB}_2$: The only difference between the hybrids is that, in 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.

$\text{HYB}_2 \overset{c}{\approx} \text{HYB}_3$: The only difference between the hybrids is that, in 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.

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

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 $\lambda_w^2 = y \oplus \lambda_w^1 \oplus \lambda_w^3 \oplus \lambda_w^4$ 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.**??**) 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_w^j for each input wire w belonging to P_5^*. For $j \notin \mathcal{S}_1$, broadcast λ_w^j for each input wire w belonging to P_1^* 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 $g' \in \{2,3,4\} \setminus \{i\}$ and $\beta \notin \mathcal{S}_g$. 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_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 23: Simulator $\mathcal{S}^{15}_{\mathsf{god5PC}}$ for $\mathsf{god5PC}$ with actively corrupt P_1^*, P_5^*

$\text{HYB}_3 \overset{c}{\approx} \text{HYB}_4$: The only difference between the hybrids is that in 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 HYB_3, Rounds 1, 2 are run as per Garble_4, which gives GC. In 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.

$\text{HYB}_4 \overset{c}{\approx} \text{HYB}_5$: 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.

$\text{HYB}_5 \overset{c}{\approx} \text{HYB}_6$: 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, $\text{IDEAL}_{\mathcal{F}_{\mathsf{god}},\mathcal{S}_{\mathsf{god3PC}}} \overset{c}{\approx} \text{REAL}_{\mathsf{god3PC},\mathcal{A}}$ [BJPR18], indistinguishability follows. □

F 3PC with GOD

In this section, we include the robust 3PC instantiation of [BJPR18] in Fig 24 for completeness. For every case of conflict when a 3PC committee is chosen, the routine $\mathsf{god3PC}$ invokes the protocol in Fig 24 to compute the output robustly while ensuring consistency of inputs committed in $\mathsf{inputGOD}$ routine. In the protocol $\mathsf{g3PC}$, 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] 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.

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 $\mathbf{Y}$ 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 25, 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 $\mathsf{O}[x^{12}]$

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 $(\texttt{conflict}, P_3)$ 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 $b \in \{0,1\}$, $(c^b_\alpha, o^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p^\alpha_1 \oplus b}_\alpha)$, $(c^b_{\ell+\alpha}, o^b_{\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p^\alpha_2 \oplus b}_{\ell+\alpha})$ 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 $\mathcal{B}_i = \left\{\mathbf{C}, \{c^b_\alpha\}_{\alpha\in[4\ell], b\in\{0,1\}}\right\}$.
- 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 $\{o^{m^\alpha_1}_\alpha, o^{x^\alpha_{31}}_{2\ell+\alpha}\}_{\alpha\in[\ell]}$, 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 $\{o^{m^\alpha_2}_{\ell+\alpha}, o^{x^\alpha_{32}}_{3\ell+\alpha}\}_{\alpha\in[\ell]}$, 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 $\mathbf{X}^\alpha_1 = \mathsf{Open}(\mathsf{pp}, c^{m^\alpha_1}_\alpha, o^{m^\alpha_1}_\alpha)$ and $\mathbf{X}^\alpha_{31} = \mathsf{Open}(\mathsf{pp}, c^{x^\alpha_{31}}_{2\ell+\alpha}, o^{x^\alpha_{31}}_{2\ell+\alpha})$ for $\alpha \in [\ell]$. Broadcast $(\texttt{conflict}, P_1)$ if Open results in $\perp$
- Assign $\mathbf{X}^\alpha_2 = \mathsf{Open}(\mathsf{pp}, c^{m^\alpha_2}_{\ell+\alpha}, o^{m^\alpha_2}_{\ell+\alpha})$, $\mathbf{X}^\alpha_{32} = \mathsf{Open}(\mathsf{pp}, c^{x^\alpha_{32}}_{3\ell+\alpha}, o^{x^\alpha_{32}}_{3\ell+\alpha})$ for $\alpha \in [\ell]$. 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 24: Protocol g3PC

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

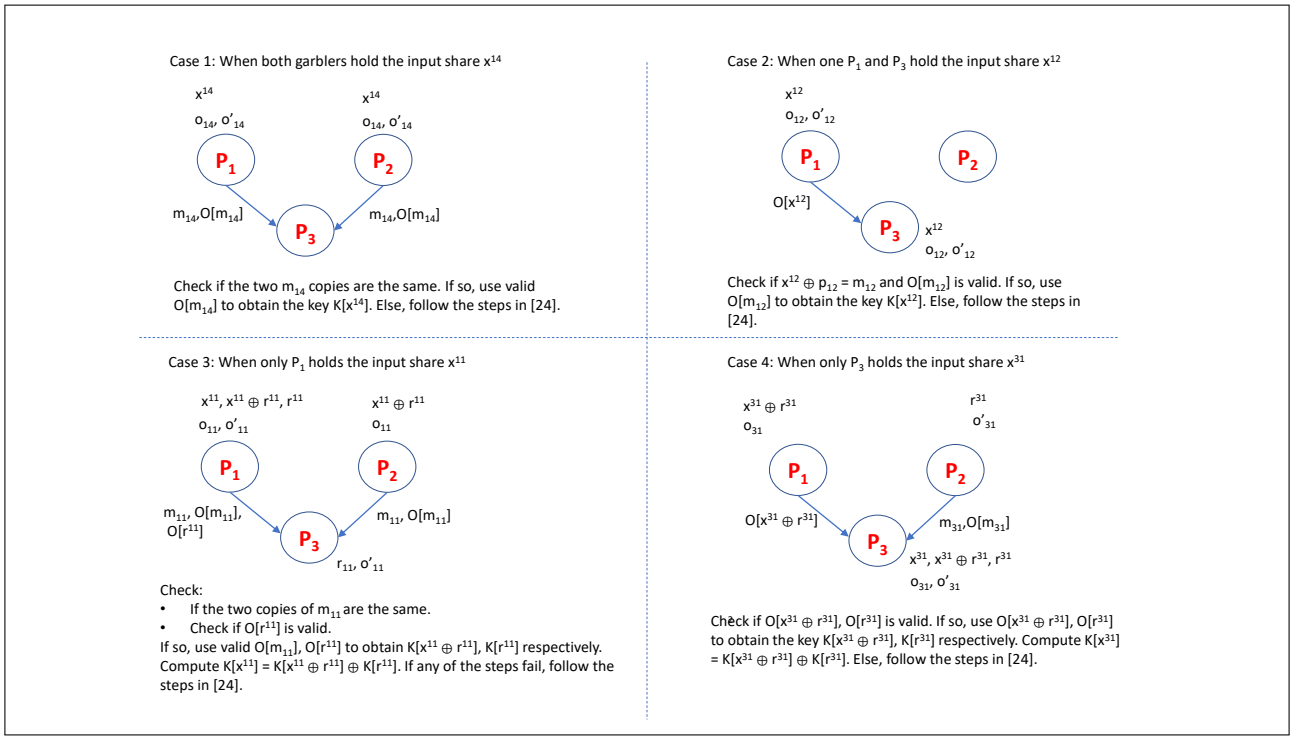

Figure 25: Diagram showing the transition from 5PC to 3PC.

the output to the garblers. Finally, the 3PC communicates the output to all the parties in 5PC. This completes the description.