

# On the Exact Round Complexity of Secure Three-Party Computation

Arpita Patra and Divya Ravi

Indian Institute of Science, India  
{arpita,divyar}@iisc.ac.in

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

In the minimal setting of pairwise-private channels, 3PC with selective abort is known to be feasible in just two rounds, while guaranteed output delivery is infeasible to achieve irrespective of the number of rounds. Settling the quest for exact round complexity of 3PC in this setting, we show that three rounds are necessary and sufficient for unanimous abort and fairness. Extending our study to the setting with an additional broadcast channel, we show that while unanimous abort is achievable in just two rounds, three rounds are necessary and sufficient for fairness and guaranteed output delivery. Our lower bound results extend for any number of parties in honest majority setting and imply tightness of several known constructions.

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

## 1 Introduction

In secure multi-party computation (MPC) [GMW87, CDG87, Yao82],  $n$  parties wish to jointly perform a computation on their private inputs in a secure way, so that no adversary  $\mathcal{A}$  actively corrupting a coalition of  $t$  parties can learn more information than their outputs (*privacy*), nor can they affect the outputs of the computation other than by choosing their own inputs (*correctness*). MPC has been a subject of extensive research and has traditionally been divided into two classes: MPC with dishonest majority [GMW87, DO10, BDOZ11, DPSZ12, GGHR14, BHP17, ACJ17] and MPC with honest majority [BGW88, CCD88, RB89, BMR90, Bea91, DN07, BFO12, BH06, CDD<sup>+</sup>99].

While the special case of MPC with dishonest majority, namely the two-party computation (2PC) has been at the focus of numerous works [Yao82, IPS08, Lin13, SS13, HKK<sup>+</sup>14, AMPR14, RR16, MR17], the same is not quite true for the special case of MPC protocols with honest majority.

The three-party computation (3PC) and MPC with small number of parties maintaining an honest majority make a fascinating area of research due to myriad reasons as highlighted below. First, they present useful use-cases in practice, as it seems that the most likely scenarios for secure MPC in practice would involve a small number of parties. In fact, the first large scale implementation of secure MPC, namely the Danish sugar beet auction [BCD<sup>+</sup>09] was designed for the three-party setting. Several other applications solved via 3PC include statistical data analysis [BTW12], email-filtering [LADM14], financial data analysis [BTW12] and distributed credential encryption service [MRZ15]. The practical efficiency of 3PC has thus got considerable emphasis in the past and some of them have evolved to technologies [Gei07, BLW08, LDDA12, LADM14, CMF<sup>+</sup>14, FLNW17, AFL<sup>+</sup>16]. Second, in practical deployments of secure computation between multiple servers that may involve long-term sensitive information, three or more servers are preferred as opposed to two. This enables recovery from faults in case one of the servers malfunctions. Third and importantly, practical applications usually demand strong security goals such as fairness (corrupted parties receive their output only if all honest parties receive output) and guaranteed output delivery (corrupted parties cannot prevent honest parties from receiving their output) which are feasible *only* in honest majority setting [Cle86]. Fourth and interestingly, there are evidences galore that having to handle a single corrupt party can be leveraged conveniently and taken advantage of to circumvent known lower bounds and impossibility results. A lower bound of three rounds has been proven in [GIKR02] for *fair* MPC with  $t \geq 2$  and arbitrary number of parties, even in the presence of broadcast channels. [IKKP15] circumvents the lower bound by presenting a *two-round* 4PC protocol tolerating a *single* corrupt party that provides guaranteed output delivery without even requiring a broadcast channel. Verifiable secret sharing (VSS) which serves as an important tool in constructing MPC protocols are known to be impossible with  $t \geq 2$  with one round in the sharing phase irrespective of the computational power of the adversary [GIKR01, PCRR09, BKP11]. Interestingly enough, a perfect VSS with  $(n = 5, t = 1)$  [GIKR01], statistical VSS with  $(n = 4, t = 1)$  [PCRR09, IKKP15] and cryptographic VSS with  $(n = 4, t = 1)$  [BKP11] are shown to be achievable with one round in the sharing phase.

The world of MPC for small population in honest majority setting witnesses a few more interesting phenomena. Assumption-wise, MPC with 3, 4 and 5 parties can be built from just One-way functions (OWF) or injective one-way functions/permutations [IKKP15, MRZ15, CGMV17], shunning public-key primitives such as Oblivious Transfer (OT) entirely, which is the primary building block in the 2-party setting. Last but not the least, the known constructions for small population in the honest majority setting perform arguably better than the constructions with two parties while offering the same level of security. For instance, 3PC with honest majority [IKKP15, MRZ15] allows to circumvent certain inherent challenges in malicious 2PC such as enforcing correctness of garbling which incurs additional communication. The exact round complex-

ity is yet another measure that sets apart the protocols with three parties over the ones with two parties. For instance, 3PC protocol is achievable just in two rounds with the minimal network setting of pairwise-private channels [IKKP15]. The 2PC (and MPC with dishonest majority) protocols achieving the same level of security (with abort) necessarily require 4 rounds [KO04] and have to resort to a common reference string (CRS) to shoot for the best possible round complexity of 2 [HLP11].

With the impressive list of motivations that are interesting from both the theoretical and practical viewpoint, we explore 3PC in the honest majority setting tolerating a malicious adversary. In this work, we set our focus on the exact round complexity of 3PC. To set the stage for our contributions, we start with a set of relevant works below.

*Related Works.* Since round complexity is considered an important measure of efficiency of MPC protocols, there is a rich body of work studying the round complexity of secure 2PC and MPC protocols under various adversarial settings and computational models. We highlight some of them below. Firstly, it is known that two rounds of interaction are essential for realizing an MPC protocol irrespective of the setting. This is because in a 1-round protocol, a corrupted party could repeatedly evaluate the “residual function” with the inputs of the honest parties fixed on many different inputs of its own (referred as “residual function” attack) [HLP11]. In the plain model, any actively secure 2PC is known to require 5 rounds in non-simultaneous message model [KO04] (under black-box simulation). The bound can be improved to 4 even in the dishonest majority setting [GMPP16] in simultaneous message model and tight upper bounds are presented in [BHP17, ACJ17, HHPV17]. With a common reference string (CRS), the lower bound can be further improved to 2 rounds [HLP11]. Tight upper bounds are shown in [GGHR14] under indistinguishability obfuscation (assumption weakened to witness encryption by [GLS15]), and in [MW16] under a variant of Fully Homomorphic Encryption (FHE) and Non-interactive Zero-knowledge.

In the honest majority setting which is shown to be necessary [Cle86] and sufficient [BGW88, CCD88, CL14] for the feasibility of protocols with fairness and guaranteed output delivery, the study on round complexity has seen the following interesting results. Three is shown to be the lower bound for fair protocols in the stand-alone model (surprisingly even with access to a CRS), assuming *non-private* channels [GLS15]. The same work presents a matching upper bound that provides guaranteed output delivery, uses a CRS and a broadcast channel and relies on a ‘special’ FHE. Their protocol can be collapsed to two rounds given access to PKI where the infrastructure carries the public keys corresponding to the ‘special’ FHE. In the plain model, three rounds are shown to be necessary for MPC with fairness and  $t \geq 2$ , even in the presence of a broadcast channel and arbitrary number of parties [GIKR02]. In an interesting work, [IKKP15] circumvents the above result by considering 4PC with *one* corruption. The protocol provides guaranteed output delivery, yet does not use a broadcast channel. In the same setting (plain model and no broadcast), [IKKP15] presents a 2-round 3PC protocol tolerating single corruption; whose communication and computation efficiency was improved by the 3-round protocol of [MRZ15]. Both these protocols achieve a weaker notion of security known as security with selective abort. Selective abort security [IKP10] (referred as ‘security with abort and no fairness’ in [GL02]) allows the corrupt parties to selectively deprive some of the honest parties of the output. In the mildly stronger

version of unanimous abort (referred as ‘security with unanimous abort and no fairness’ in [GL02]), either all or none of the honest parties receive the output. An easy observation concludes that the 3PC of [MRZ15] achieves unanimous abort, when its third round message is broadcasted, albeit for functions giving the same output to all. The works relevant to honest majority setting are listed below.

3PC has been studied in different settings as well. High-throughput MPC with non-constant round complexity are studied in [FLNW17, AFL<sup>+</sup>16]. [CKMZ14] studies 3PC with dishonest majority. Recently, [CGMV17] presents a practically efficient 5-party MPC protocol in honest majority setting, going beyond 3-party case, relying on distributed garbling technique based on [BMR90].

Ref.	Setting	Round	Network Setting / Assumption	Security	Comments
[AJL <sup>+</sup> 12]	$t < n/2$	$\geq 5$	private channel, Broadcast / CRS, FHE, NIZK	fairness	upper bound
[GLS15]	$t < n/2$	3	non-private channel, Broadcast / CRS, FHE	guaranteed output delivery	upper bound
[GLS15]	$t < n/2$	2	non-private channel, Broadcast / CRS, PKI, FHE	guaranteed output delivery	upper bound
[IKP10]	$n = 5, t = 1$	2	private channel / OWF	guaranteed output delivery	upper bound
[IKKP15]	$n = 3, t = 1$	2	private channel / OWF	selective abort	upper bound
[IKKP15]	$n = 4, t = 1$	2	private channel / (injective) OWF	guaranteed output delivery	upper bound
[MRZ15]	$n = 3, t = 1$	3	private channel, Broadcast / PRG	unanimous abort	upper bound
[GLS15]	$t < n/2$	3	non-private channel, Broadcast / CRS	fairness	lower bound
[GIKR02]	$n; t > 1$	3	private channel, Broadcast	fairness	lower bound

## 1.1 Our Results

In this paper, we set our focus on the exact round complexity of 3PC protocols with one active corruption achieving a range of security notions, namely selective abort, unanimous abort, fairness and guaranteed output delivery in a setting with pair-wise private channels and without or with a broadcast channel (and no additional setup). In the minimal setting of pair-wise private channels, it is known that 3PC with selective abort is feasible in just two rounds [IKKP15], while guaranteed output delivery is infeasible to achieve irrespective of the number of rounds [CHOR16]. No bound on round complexity is known for unanimous abort or fairness. In the setting with a broadcast channel, the result of [MRZ15] implies 3-round 3PC with unanimous abort. Neither the round optimality of the [MRZ15] construction, nor any bound on round complexity is known for protocols with fairness and guaranteed output delivery.

This work settles all the above questions via two lower bound results and three upper bounds. Both our lower-bounds extend for general  $n$  and  $t$  with strict honest majority i.e.  $n/3 \leq t < n/2$ . They imply tightness of several known constructions of [IKKP15] and complement the lower bound of [GIKR02] which holds for only  $t > 1$ . Our upper bounds are from injective (one-to-one) one-way functions. The fundamental concept of garbled circuits (GC) contributes as their key basis, following several prior works in this domain [CKMZ14, IKKP15, MRZ15]. The techniques in our upper bounds do not seem to extend for  $t > 1$ , leaving open designing round-optimal protocols for the general case with various security notions. We now elaborate on the results below:

*Without Broadcast Channel.* In this paper, we show that three rounds are necessary to achieve 3PC with unanimous abort and fairness, in the absence of a broadcast channel. The sufficiency is proved via a 3-round fair protocol (which also achieves unanimous

abort security). Our lower bound result immediately implies tightness of the 3PC protocol of [IKKP15] achieving selective abort in two rounds, in terms of security achieved. This completely settles the questions on exact round complexity of 3PC in the minimal setting of pair-wise private channels. Our 3-round fair protocol uses a sub-protocol that is reminiscent of Conditional Disclosure of Secrets (CDS) [GIKM00], with an additional property of authenticity that allows a recipient to detect the correct secret. Our implementation suggests a realisation of authenticated CDS from privacy-free GCs.

*With Broadcast Channel.* With access to a broadcast channel, we show that it takes just two rounds to get 3PC with unanimous abort, implying non-optimality of the 3-round construction of [MRZ15]. On the other hand, we show that three rounds are necessary to construct a 3PC protocol with fairness and guaranteed output delivery. The sufficiency for fairness already follows from our 3-round fair protocol without broadcast. The sufficiency for guaranteed output delivery is shown via yet another construction in the presence of broadcast. The lower bound result restricted for  $t = 1$  complements the lower bound of [GIKR02] making three rounds necessary for MPC with fairness in the honest majority setting for all the values of  $t$ . The lower bound further implies that for two-round fair (or guaranteed output delivery) protocols with one corruption, the number of parties needs to be at least four, making the 4PC protocol of [IKKP15] an optimal one. Notably, our result does not contradict with the two-round protocol of [GLS15] that assumes PKI (where the infrastructure contains the public keys of a ‘special’ FHE), CRS and also broadcast channel.

The table below captures the complete picture of the round complexity of 3PC. The necessity of two rounds for any type of security follows from [HLP11] via the ‘residual attack’. Notably, broadcast facility only impacts the round complexity of unanimous abort and guaranteed output delivery, leaving the round complexity of selective abort and fairness unperturbed.

Security	Without Broadcast	References Necessity/Sufficiency	With Broadcast	References Necessity/Sufficiency
Selective Abort	2	[HLP11] / [IKKP15]	2	[HLP11] / [IKKP15]
Unanimous Abort	3	<b>This paper / This paper</b>	2	[HLP11] / <b>This paper</b>
Fairness	3	<b>This paper / This paper</b>	3	<b>This paper / This paper</b>
Guaranteed output delivery	Impossible	[CHOR16]	3	<b>This paper / This paper</b>

## 1.2 Techniques

*Lower Bounds.* We present two lower bounds– **(a)** three rounds are necessary for achieving fairness in the presence of pair-wise channels and a broadcast channel; **(b)** three rounds are necessary for achieving unanimous abort in the presence of just pair-wise channels. The lower bounds are shown by taking a special 3-party function and by devising a sequence hybrid executions under different adversarial strategies allowing to conclude any 3PC protocol computing the considered function cannot be private. Gliding over the hybrids uses the fact that the protocol is fair or unanimous abort in the respective proofs.

*Upper Bounds.* We present three upper bounds– **(a)** 3-round fair protocol; **(b)** 2-round protocol with unanimous abort and **(c)** 3-round protocol with

guaranteed output delivery. The former in the presence of just pairwise channels, the latter two with an additional broadcast channel. The known generic transformations such as, unanimous abort to (identifiable) fairness [IKP<sup>+</sup>16] or identifiable fairness to guaranteed output delivery [CL14], does not help in any of our constructions. For instance, any 3-round fair protocol without broadcast cannot take the former route as it is not round-preserving and unanimous abort in two rounds necessarily requires broadcast as shown in this work. A 3-round protocol with guaranteed output delivery cannot be constructed combining both the transformations due to inflation in round complexity.

Building on the protocol of [MRZ15], the basic building block of our protocols needs two of the parties to enact the role of the garbler and the remaining party to carry out the responsibility of circuit evaluation. Constrained with just two or three rounds, our protocols are built from the parallel composition of three sub-protocols, each one with different party enacting the role of the evaluator (much like [IKKP15]). Each sub-protocol consumes two rounds. Based on the security needed, the sub-protocols deliver distinct flavours of security with ‘identifiable abort’. For the fair and unanimous abort, the identifiability is in the form of conflict that is local (privately known) and public/global (known to all) respectively, while for the protocol with guaranteed output delivery, it is local identification of the corrupt. Achieving such identifiability in just two rounds (sometime without broadcast) is challenging in themselves. Pulling up the security guarantee of these subprotocols via entwining three executions to obtain the final goals of fairness, unanimous abort and guaranteed output delivery constitute yet another novelty of this work. Maintaining the input consistency across the three executions pose another challenge that are tackled via mix of novel techniques (that consume no additional cost in terms of communication) and existing tricks such as ‘proof-of-cheating’ or ‘cheat-recovery’ mechanism [Lin13, CKMZ14]. The issue of input consistency does not appear in the construction of [MRZ15] at all, as it does not deal with parallel composition. On the other hand, the generic input consistency technique adopted in [IKKP15] can only (at the best) detect a conflict locally and cannot be extended to support the stronger form of identifiability that we need.

Below, we present the common issues faced and approach taken in all our protocols before turning towards the challenges and way-outs specific to our constructions. Two of the major efficiency bottlenecks of 2PC from garbled circuits, namely the need of multiple garbled circuits due to cut-and-choose approach and Oblivious Transfer (OT) for enabling the evaluator to receive its input in encoded form are bypassed in the 3PC scenario through two simple tricks [IKKP15, MRZ15]. First, the garblers use common randomness to construct the same garbled circuit individually. A simple comparison of the GCs received from the two garblers allows to conclude the correctness of the GC. Since at most one party can be corrupt, if the received GCs match, then its correctness can be concluded. Second, the evaluator shares its input additively among the garblers at the onset of the protocol, reducing the problem to a secure computation of a function on the garblers’ inputs alone. Specifically, assuming  $P_3$  as the evaluator, the computation now takes inputs from  $P_1$  and  $P_2$  as  $(x_1, x_{31})$  and  $(x_2, x_{32})$  respectively to compute  $C(x_1, x_2, x_{31}, x_{32}) = f(x_1, x_2, x_{31} \oplus x_{32})$ . Since the garblers possess all the inputs needed for the computation, OT is no longer needed to transfer the evaluator’s input in encoded form to  $P_3$ .

Next, to force the garblers to input encoding and decoding information (the keys) that are consistent with the GCs, the following technique is adopted. Notice that the issue of input consistency where a corrupt party may use different inputs as an evaluator and as a garbler in different instances of the sub-protocols is distinct and remains to be tackled separately. Together with the GC, each garbler also generates the commitment to the encoding and decoding information using the common shared randomness and communicates to the evaluator. Again a simple check on whether the set of commitments are same for both the garblers allows to conclude their correctness. Now it is infeasible for the garblers to decommit the encoded input corresponding to their own input and the evaluator's share to something that are inconsistent to the GC without being caught. Following a common trick to hide the inputs of the garblers, the commitments on the encoding information corresponding to every bit of the garblers' input are sent in permuted order that is privy to the garblers. The commitment on the decoding information is relevant only for the fair protocol where the decoding information is withheld to force a corrupt evaluator to be fair. Namely, in the third round of the final protocol, the evaluator is given access to the decoding information only when it helps the honest parties to compute the output. This step needs us to rely on the obliviousness of our garbling scheme, apart from privacy. The commitment on the decoding information and its verification by crosschecking across the garblers are needed to prevent a corrupt party to lie later. Now we turn to the challenges specific to the constructions.

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

A party is rewarded with a 'certificate' for enacting an honest sender and emulating a broadcast by sending the same information to the other two parties, for the common information such as GCs and commitments. This protocol internally mimics a CDS protocol [GIKM00] for equality predicate, with an additional property of 'authenticity', a departure from the traditional CDS. An authenticated CDS allows the receiver to detect correct receipt of the secret/certificate (similar to authenticated encryption where the receiver knows if the received message is the desired one). As demonstrated below, the certificate allows to identify the culprit behind the confusion on one hand, and to securely transmit the decoding information from a confused honest party to the happy honest party in the third round, on the other. The certificate, being a proof of correct behaviour, when comes from an honest party, say  $P_i$ , the other honest party who sees conflict in the information distributed by  $P_i$  communicated over point-to-point channel, can readily identify the corrupt party responsible for creating the conflict in Round 3.

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

Yet again, we use garbled circuits to implement the above where a party willing to receive a certificate acts as an evaluator for a garbled circuit implementing ‘equality’ check of the inputs. The other two parties act as the garblers with their inputs as the common information dealt by the evaluator. With no concern of input privacy, the circuit can be garbled in a privacy-free way [JKO13, FNO15]. The certificate that is the key for output 1 is accessible to the evaluator only when it emulates a broadcast by dealing identical copies of the common information to both the other parties. Notably, [IW14] suggests application of garbling to realise CDS.

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

*Achieving guaranteed output delivery in 3 rounds.* For achieving this stronger notion, the sub-protocol here needs a stronger kind of identifiability, identifying the corrupt locally to be specific, to facilitate all parties to get output within an additional round no matter what. To this effect, our sub-protocol is enhanced so that the evaluator either successfully computes the output or identifies the corrupt party. We emphasise that the goals of the sub-protocols for unanimous abort and guaranteed output delivery, namely global conflict vs. local identification, are orthogonal and do not imply each other. The



additional challenge faced in composing the executions to achieve guaranteed output delivery lies in determining the appropriate ‘committed’ input of the corrupt party based on which round and execution of sub-protocol it chooses to strike.

*Tackling input consistency.* We take a uniform approach for all our protocols. We note that a party takes three different roles across the three composed execution: an evaluator, a garbler who initiate the GC generation by picking the randomness, a co-garbler who verifies the sanity of the GC. In each instance, it gets a chance to give inputs. We take care of input consistency in two parts. First, we tie the inputs that a party can feed as an evaluator and as a garbler who initiates a GC construction via a mechanism that needs no additional communication at all. This is done by setting the permutation strings (used to permute the commitments of encoding information of the garblers) to the shares of these parties’ input in a certain way. The same trick fails to work in two rounds for the case when a party acts as a garbler and a co-garbler in two different executions. We tackle this by superimposing two mirrored copies of the sub-protocol where the garblers exchange their roles. Namely, in the final sub-protocol, each garbler initiates an independent copy of garbled circuit and passes on the randomness used to the fellow garbler for verification. The previous trick is used to tie the inputs that a party feeds as an evaluator and as a garbler for the GC initiated by it (inter-execution consistency). The input consistency of a garbler for the two garbled circuits (one initiated by him and the other by the co-garbler) is taken care using ‘proof-of-cheating’ mechanism [Lin13] where the evaluator can unlock the clear input of both the other parties using conflicting output wire keys (intra-execution consistency). While this works for our protocols with unanimous abort and guaranteed output delivery, the fair protocol faces additional challenges. First, based on whether a party releases a clear or encoded input, a corrupt garbler feeding two different inputs can conclude whether  $f$  leads to the same output for both his inputs, breaching privacy. This is tackled by creating the ciphertexts using conflicting input keys. Second, inspite of the above change, a corrupt garbler can launch ‘selective failure attack’ [MF06, KS06] and breach privacy of his honest co-garbler. We tackle this using ‘XOR-tree approach’ [LP07] where every input bit is broken into  $s$  shares and security is guaranteed except with probability  $2^{-(s-1)}$  per input bit. We do not go for the refined version of this technique, known as probe-resistant matrix, [LP07, SS13] for simplicity.

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

*On concrete efficiency.* Though the focus is on the round complexity, the concrete efficiency of our protocols is comparable to Yao [Yao82] and require transmission and evaluation of few GCs (upto 9) (in some cases we only need privacy-free GCs which permit more efficient constructions than their private counterparts [JKO13, FNO15]). The broadcast communication of the optimized variants of our protocols is independent of the GC size via applying hash function. We would like to draw attention towards the new tricks such as the ones used for input consistency, getting certificate of good behaviour via garbled circuits, which may be of both theoretical and practical interest.

We believe the detailed take on our protocols will help to lift them or their derivatives to practice in future.

### 1.3 Roadmap

We present a high-level overview of the primitives used in Section 2. The security definition and the functionalities appear in Appendix B. We present our 3-round fair protocol, 2-round protocol with unanimous abort and 3-round protocol with guaranteed output delivery in Section 3, 4 and 5 respectively. The respective security proofs appear in Appendices D, E and F and the common optimizations in Appendix C. Our lower bound results appear in Section 6. We define authenticated CDS in Appendix G and show its realisation from one of the sub-protocol used in our 3-round fair protocol.

## 2 Preliminaries

### 2.1 Model

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

We denote the cryptographic security parameter by  $\kappa$ . A negligible function in  $\kappa$  is denoted by  $\text{negl}(\kappa)$ . A function  $\text{negl}(\cdot)$  is *negligible* if for every polynomial  $p(\cdot)$  there exists a value  $N$  such that for all  $m > N$  it holds that  $\text{negl}(m) < \frac{1}{p(m)}$ . We denote by  $[x]$ , the set of elements  $\{1, \dots, x\}$  and by  $[x, y]$  for  $y > x$ , the set of elements  $\{x, x + 1, \dots, y\}$ . For any  $x \in_R \{0, 1\}^m$ ,  $x^i$  denotes the bit of  $x$  at index  $i$  for  $i \in [m]$ . Let  $S$  be an infinite set and  $X = \{X_s\}_{s \in S}, Y = \{Y_s\}_{s \in S}$  be distribution ensembles. We say  $X$  and  $Y$  are computationally indistinguishable, if for any PPT distinguisher  $\mathcal{D}$  and all sufficiently large  $s \in S$ , we have  $|\Pr[\mathcal{D}(X_s) = 1] - \Pr[\mathcal{D}(Y_s) = 1]| < 1/p(|s|)$  for every polynomial  $p(\cdot)$ .

### 2.2 Primitives

*Garbling Schemes.* The term ‘garbled circuit’ (GC) was coined by Beaver [BMR90], but it had largely only been a technique used in secure protocols until they were formalized as a primitive by Bellare et al. [BHR12]. ‘Garbling Schemes’ as they were termed, were assigned well-defined notions of security, namely *correctness*, *privacy*, *obliviousness*, and *authenticity*. A garbling scheme  $\mathcal{G}$  is characterised by a tuple of PPT algorithms  $\mathcal{G} = (\text{Gb}, \text{En}, \text{Ev}, \text{De})$  described below.

- $\text{Gb}(1^\kappa, C)$  is invoked on a circuit  $C$  in order to produce a ‘garbled circuit’  $C$ , ‘input encoding information’  $e$ , and ‘output decoding information’  $d$ .

- $\text{En}(x, e)$  encodes a clear input  $x$  with encoding information  $e$  in order to produce a garbled/encoded input  $\mathbf{X}$ .
- $\text{Ev}(\mathbf{C}, \mathbf{X})$  evaluates  $\mathbf{C}$  on  $\mathbf{X}$  to produce a garbled/encoded output  $\mathbf{Y}$ .
- $\text{De}(\mathbf{Y}, d)$  translates  $\mathbf{Y}$  into a clear output  $y$  as per decoding information  $d$ .

We give an informal intuition of the notion captured by each of the security properties, namely *correctness*, *privacy*, *obliviousness*, and *authenticity*. Correctness enforces that a correctly garbled circuit, when evaluated, outputs the correct output of the underlying circuit. Privacy aims to protect the privacy of encoded inputs. Authenticity enforces that the evaluator can only learn the output label that corresponds to the value of the function. Obliviousness captures the notion that when the decoding information is withheld, the garbled circuit evaluation leaks no information about *any* underlying clear values; be they of the input, intermediate, or output wires of the circuit. The formal definitions are deferred to Appendix A.1.

We are interested in a class of garbling schemes referred to as *projective* in [BHR12]. When garbling a circuit  $C : \{0, 1\}^n \mapsto \{0, 1\}^m$ , a projective garbling scheme produces encoding information of the form  $e = (e_i^0, e_i^1)_{i \in [n]}$ , and the encoded input  $\mathbf{X}$  for  $x = (x_i)_{i \in [n]}$  can be interpreted as  $\mathbf{X} = \text{En}(x, e) = (e_i^{x_i})_{i \in [n]}$ .

Our 3-round fair protocol relies on garbling schemes that are simultaneously correct, private and oblivious. One of its subroutine uses a garbling scheme that is only authentic. Such schemes are referred as *privacy-free* [JKO13, FNO15]. Our protocols with unanimous abort and guaranteed output delivery need a correct, private and authentic garbling scheme that need not provide obliviousness. Both these protocols as well as the privacy-free garbling used in the fair protocol further need an additional decoding mechanism denoted as *soft decoding* algorithm  $\text{sDe}$  [MRZ15] that can decode garbled outputs without the decoding information  $d$ . The soft-decoding algorithm must comply with correctness:  $\text{sDe}(\text{Ev}(\mathbf{C}, \text{En}(e, x))) = C(x)$  for all  $(\mathbf{C}, e, d)$ . While both  $\text{sDe}$  and  $\text{De}$  can decode garbled outputs, the authenticity needs to hold only with respect to  $\text{De}$ . In practice, soft decoding in typical garbling schemes can be achieved by simply appending the truth value to each output wire label.

*Non-interactive Commitment Schemes.* A non-interactive commitment scheme (NICOM) consists of two algorithms (Com, Open) defined as follows. Given a security parameter  $\kappa$ , a common parameter  $\text{pp}$ , message  $x$  and random coins  $r$ , PPT algorithm Com outputs commitment  $c$  and corresponding opening information  $o$ . Given  $\kappa$ ,  $\text{pp}$ , a commitment and corresponding opening information  $(c, o)$ , PPT algorithm Open outputs the message  $x$ . The algorithms should satisfy correctness, binding (i.e. it must be hard for an adversary to come up with two different openings of any  $c$  and *any*  $\text{pp}$ ) and hiding (a commitment must not leak information about the underlying message) properties. We need this kind of strong binding as the same party who generates the  $\text{pp}$  and commitment is required to open later. Two such instantiations of NICOM based on symmetric key primitives (specifically, injective one-way functions) and the formal definitions of the properties are given in Appendix A.2.

We also need a NICOM scheme that admits equivocation property. An equivocal non-interactive commitment (eNICOM) is a NICOM that allows equivocation of a cer-

tain commitment to any given message with the help of a trapdoor. The formal definitions and instantiations appear in Appendix A.3.

*Symmetric-Key Encryption (SKE) with Special Correctness.* Our fair protocol uses a SKE  $\pi = (\text{Gen}, \text{Enc}, \text{Dec})$  which satisfies CPA security and a special correctness property [JW16, LP09]– if the encryption and decryption keys are different, then decryption fails with high probability. The definition and an instantiation appear in Appendix A.4.

### 3 3-round 3PC with Fairness

This section presents a tight upper bound for 3PC achieving fairness in the setting with just pair-wise private channels. Our result from Section 6.2 rules out the possibility of achieving fairness in 2 rounds in the same setting. Our result from Section 6.1 further shows tightness of 3 rounds even in the presence of a broadcast channel.

Building on the intuition given in the introduction, we proceed towards more detailed discussion of our protocol. Our fair protocol is built from parallel composition of three copies of each of the following two sub-protocols: (a)  $\text{fair}_i$  where  $P_i$  acts as the evaluator and the other two as garblers for computing the desired function  $f$ . This sub-protocol ensures that honest  $P_i$  either computes its encoded output or identifies just a conflict in the worst case. The decoding information is committed to  $P_i$ , yet not opened. It is released in Round 3 of the final composed protocol under subtle conditions as elaborated below. (b)  $\text{cert}_i$  where  $P_i$  acts as the evaluator and the other two as garblers for computing an equality checking circuit on the common information distributed by  $P_i$  in the first round of the final protocol. Notably, though the inputs come solely from the garblers, they are originated from the evaluator and so the circuit can be garbled in a privacy-free fashion. This sub-protocol ensures either honest  $P_i$  gets its certificate, the key for output 1 (meaning the equality check passes through), or identifies a conflict in the worst case. The second round of  $\text{cert}_i$  is essentially an ‘authenticated’ CDS for equality predicate tolerating one active corruption as discussed in Appendix G. Three *global* variables are maintained by each party  $P_i$  to keep tab on the conflicts and the corrupt. Namely,  $\mathcal{C}_i$  to keep the identity of the corrupt,  $\text{flag}_j$  and  $\text{flag}_k$  (for distinct  $i, j, k \in [3]$ ) as indicators of detection of conflict with respect to information distributed by  $P_j$  and  $P_k$  respectively. The sub-protocols  $\text{fair}_i$  and  $\text{cert}_i$  assure that if neither the two flags nor  $\mathcal{C}_i$  is set, then  $P_i$  must be able to evaluate the GC successfully and get its certificate respectively.

Once  $\{\text{fair}_i, \text{cert}_i\}_{i \in [3]}$  complete by the end of round 2 of the final protocol fair, any honest party will be in one of the three states: (a) no corruption and no conflict detected ( $(\mathcal{C}_i = \emptyset) \wedge (\text{flag}_j = 0) \wedge (\text{flag}_k = 0)$ ); (b) corruption detected ( $\mathcal{C}_i \neq \emptyset$ ); (c) conflict detected ( $\text{flag}_j = 1) \vee (\text{flag}_k = 1)$ ). An honest party, guaranteed to have computed its encoded output and certificate *only* in the first state, releases these as well as the decoding information for both the other parties unconditionally in the third round. In the other two states, an honest party conditionally releases only the decoding information. This step is extremely crucial for maintaining fairness. Specifically, a party that belongs to the second state, releases the decoding information only to the party identified to be honest. A party that belongs to the third state, releases the decoding

information in encrypted form *only* to the party whose distributed information are not agreed upon, so that the encryption can be unlocked only via a valid certificate. A corrupt party will either have its certificate or the encrypted decoding information, but *not* both. The former when it distributes its common information correctly and the latter when it does not. The only way a corrupt party can get its decoding information is by keeping one honest party in the first state, in which case both the honest parties will be able to compute the output as follows. The honest party in state one, say  $P_i$ , either gets its decoding information on clear or in encrypted form. The former when the other honest party,  $P_j$  is in the first or second state and the latter when  $P_j$  is in the third state.  $P_i$  retrieves the decoding information no matter what, as it also holds the certificate to open the encryption. An honest party  $P_j$  in the second state, on identifying  $P_i$  as honest, takes the encoded output of  $P_i$  and uses its own decoding information to compute the output. The case for an honest party  $P_j$  in the third state is the most interesting. Since honest  $P_i$  belongs to the first state, a corrupt party must have distributed its common information correctly as otherwise  $P_i$  will find a conflict and would be in third state. Therefore,  $P_j$  in the third state must have found  $P_i$ 's information on disagreement due to the corrupt party's misbehaviour. Now,  $P_i$ 's certificate that proves his correct behaviour, allows  $P_j$  to identify the corrupt, enter into the second state and compute the output by taking the encoded output of honest  $P_i$ . In the following, we describe execution  $\text{fair}_i$  assuming input consistency, followed by  $\text{cert}_i$ . Entwining the six executions, tackling the input consistency and the final presentation of protocol  $\text{fair}$  appear in the end.

### 3.1 Protocol $\text{fair}_i$

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

To avoid issues of adaptivity, the GCs are not sent on clear in the first round to  $P_i$  who may choose its input based on the GCs. Rather, a garbler sends a commitment to its GC to  $P_i$  and it is opened only by the co-garbler after successful scrutiny. The correctness of evaluated GC still carries over as a corrupt garbler cannot open to a different circuit than the one committed by an honest garbler by virtue of the binding property of the commitment scheme. We use an eNICOM for committing the GCs and decoding information as equivocation is needed to tackle a technicality in the security proof. The

simulator of our final protocol needs to send the commitments on GC, encoding and decoding information without having access to the input of an evaluator  $P_i$  (and thus also the output), while acting on behalf of the honest garblers in  $\text{fair}_i$ . The eNICOM cannot be used for the encoding information, as they are opened by the ones who generate the commitments and eNICOM does not provide binding in such a case. Instead, the GCs and the decoding information are equivocated based on the input of the evaluator and the output.

Protocol  $\text{fair}_i$  appears in Figure 1 where  $P_i$  returns encoded outputs  $\mathbf{Y}_i = (\mathbf{Y}_i^j, \mathbf{Y}_i^k)$  (initially set to  $\perp$ ) for the circuits created by  $P_j, P_k$ , the commitments to the respective decoding information  $C_j^{\text{dec}}, C_k^{\text{dec}}$  and the flags  $\text{flag}_j, \text{flag}_k$  (initially set to false) to be used in the final protocol. The garblers output their respective corrupt set, flag for the fellow garbler and opening for the decoding information corresponding to its co-garbler's GC and *not* its own. This is to ensure that it cannot break the binding of eNICOM which may not necessarily hold for adversarially-picked public parameter.

**Lemma 1.** *During  $\text{fair}_i$ ,  $P_\beta \notin \mathcal{C}_\alpha$  holds for honest  $P_\alpha, P_\beta$ .*

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

**Lemma 2.** *If honest  $P_i$  has  $\mathcal{C}_i = \emptyset$  and  $\text{flag}_j = \text{flag}_k = 0$ , then  $\mathbf{Y}_i = (\mathbf{Y}_i^j, \mathbf{Y}_i^k) \neq \perp$ .*

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

### 3.2 Protocol $\text{cert}_i$

When a party  $P_i$  in  $\text{fair}_i$  is left in a confused state and has no clue about the corrupt, it is in dilemma on whether or whose encoded output should be used to compute output and who should it release the decoding information (that it holds as a garbler) to in the final protocol. Protocol  $\text{cert}_i$ , in a nutshell, is introduced to help a confused party to identify the corrupt and take the honest party's encoded output for output computation, on one hand, and to selectively deliver the decoding information only to the other honest party, on the other. Protocol  $\text{cert}_i$  implements evaluation of an equality checking function that takes inputs from the two garblers and outputs 1 when the test passes and outputs the inputs themselves otherwise. In the final protocol, the inputs are the common information (GCs and commitments) distributed by  $P_i$  across all executions of  $\text{fair}_j$ . The certificate is the output key corresponding to output 1. Since input privacy is not a concern here,

**Protocol fair<sub>i</sub>()**

**Inputs:** Party  $P_\alpha$  has  $x_\alpha$  for  $\alpha \in [3]$ .

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

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

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

**Round 1:**

- $P_i$  randomly secret shares his input  $x_i$  as  $x_i = x_{ij} \oplus x_{ik}$  and sends  $x_{ij}$  to  $P_j$  and  $x_{ik}$  to  $P_k$ .
- $P_l$  for  $l \in \{j, k\}$  samples  $s_l \in_R \{0, 1\}^\kappa$ ,  $\text{epp}_l$  and  $\text{pp}_l$  for  $G$ , eNICOM and NICOM resp. and:
  - o compute garbled circuit  $(\mathbf{C}_l, e_l, d_l) \leftarrow \text{Gb}(1^\kappa, C)$  using randomness from  $G(s_l)$ . Assume  $\{e_{l\alpha}^0, e_{l\alpha}^1\}_{\alpha \in [\ell]}$ ,  $\{e_{l(\ell+\alpha)}^0, e_{l(\ell+\alpha)}^1\}_{\alpha \in [\ell]}$ ,  $\{e_{l(2\ell+\alpha)}^0, e_{l(2\ell+\alpha)}^1\}_{\alpha \in [2\ell]}$  denote the encoding information for the input of  $P_j, P_k$  and the secret shares of  $P_i$  respectively.
  - o compute commitments for GC and decoding information.  $(c_l, o_l) \leftarrow \text{eCom}(\text{epp}_l, \mathbf{C}_l)$  and  $(c_l^{\text{dec}}, o_l^{\text{dec}}) \leftarrow \text{eCom}(\text{epp}_l, d_l)$ .
  - o sample permutation strings  $p_{lj}, p_{lk} \in_R \{0, 1\}^\ell$  for the inputs of  $P_j$  and  $P_k$ . Compute commitments to encoding information as: for  $b \in \{0, 1\}$ ,  $(c_{l\alpha}^b, o_{l\alpha}^b) \leftarrow \text{Com}(\text{pp}_l, e_{l\alpha}^{p_{lj} \oplus b})$ ,  $(c_{l(\ell+\alpha)}^b, o_{l(\ell+\alpha)}^b) \leftarrow \text{Com}(\text{pp}_l, e_{l(\ell+\alpha)}^{p_{lk} \oplus b})$  when  $\alpha \in [\ell]$ ,  $(c_{l(2\ell+\alpha)}^b, o_{l(2\ell+\alpha)}^b) \leftarrow \text{Com}(\text{pp}_l, e_{l(2\ell+\alpha)}^b)$  when  $\alpha \in [2\ell]$ .
  - o send  $\mathcal{D}_l = (\text{epp}_l, \text{pp}_l, c_l, \{c_{l\alpha}^b\}_{\alpha \in [4\ell], b \in \{0,1\}}, c_l^{\text{dec}})$  to both the other parties and send  $\{s_l, p_{lj}, p_{lk}, o_l, \{o_{l\alpha}^b\}_{\alpha \in [4\ell], b \in \{0,1\}}, o_l^{\text{dec}}\}$  only to co-garbler  $P_{\{j,k\} \setminus l}$ .
- $P_j$  sets  $\mathcal{C}_j = \mathcal{P}_k$  if  $\mathcal{D}_k$  and  $\{s_k, p_{kj}, p_{kk}, o_k, \{o_{k\alpha}^b\}_{\alpha \in [4\ell], b \in \{0,1\}}, o_k^{\text{dec}}\}$  are inconsistent. Else, set  $O_i^{\text{dec}} = o_k^{\text{dec}}$ .  $P_k$  performs similar steps for the values received from  $P_j$ .

**Round 2:**

- $P_i$  sends  $\mathcal{D}_j$  to  $P_k$  and  $\mathcal{D}_k$  to  $P_j$ .  $P_j$  sets  $\text{flag}_k = 1$  if  $\mathcal{D}_k$  received from  $P_i$  and  $P_k$  does not match. Similar step is executed by  $P_k$ .
  - $P_j$  computes the indicator strings  $m_{jj} = p_{jj} \oplus x_j, m_{kj} = p_{kj} \oplus x_j$  for its inputs. If  $P_k \notin \mathcal{C}_j$ , then send  $(\text{OK}, \mathcal{D}_k, (o_k, \{o_{k\alpha}^b, x_{j\alpha}^b\}_{\alpha \in [\ell]}, m_{kj}), (\{o_{j\alpha}^b, x_{j(2\ell+\alpha)}^b\}_{\alpha \in [\ell]}, m_{jj}))$  to  $P_i$ . Else, send nOK to  $P_i$ .  $P_k$  performs similar steps.
  - (Local Computation)  $P_i$  sets  $\mathbf{Y}_i^j = \perp$  and  $\text{flag}_j = 1$  when (a)  $P_k$  sent nOK or (b)  $\mathcal{D}_j$  sent by  $P_j$  and  $P_k$  do not match. Otherwise,  $P_i$  sets  $C_j^{\text{dec}} = c_j^{\text{dec}} \in \mathcal{D}_j$  and does:
    - o open  $\mathcal{C}_j \leftarrow \text{eOpen}(\text{epp}_j, c_j, o_j)$  with  $o_j$  received from  $P_k$ . Set  $\mathcal{C}_i = \mathcal{P}_k$  if  $\mathcal{C}_j = \perp$ .
    - o open  $\mathbf{X}_j^\alpha = \text{Open}(\text{pp}_j, c_{j\alpha}^{m_{jj}^\alpha}, o_{j\alpha}^{m_{jj}^\alpha})$ ,  $\mathbf{X}_{ij}^\alpha = \text{Open}(\text{pp}_j, c_{j(2\ell+\alpha)}^{x_{ij}^\alpha}, o_{j(2\ell+\alpha)}^{x_{ij}^\alpha})$ , for  $\alpha \in [\ell]$ , for the opening received from  $P_j$  and the commitments taken from  $\mathcal{D}_j$ . Include  $P_j$  in  $\mathcal{C}_i$  if any of the opened input labels above is opened to  $\perp$ .
    - o open  $\mathbf{X}_k^\alpha = \text{Open}(\text{pp}_j, c_{j(\ell+\alpha)}^{m_{jk}^\alpha}, o_{j(\ell+\alpha)}^{m_{jk}^\alpha})$  and  $\mathbf{X}_{ik}^\alpha = \text{Open}(\text{pp}_j, c_{j(3\ell+\alpha)}^{x_{ik}^\alpha}, o_{j(3\ell+\alpha)}^{x_{ik}^\alpha})$  for  $\alpha \in [\ell]$ , for the opening received from  $P_k$  and the commitments taken from  $\mathcal{D}_j$ . Include  $P_k$  in  $\mathcal{C}_i$  if any of the opened input labels above is opened to  $\perp$ .
    - o If  $\mathcal{C}_i = \emptyset$ , set  $\mathbf{X} = \mathbf{X}_j | \mathbf{X}_k | \mathbf{X}_{ij} | \mathbf{X}_{ik}$ , run  $\mathbf{Y}_i^j \leftarrow \text{Ev}(\mathcal{C}_j, \mathbf{X})$ . Else set  $\mathbf{Y}_i^j = \perp$
- Similar steps for  $\mathcal{C}_k$  will be executed to compute  $\mathbf{Y}_i^k$ , populate  $\mathcal{C}_i$  and update  $\text{flag}_k$ .

Fig. 1: Protocol fair<sub>i</sub>

the circuit is enough to be garbled in privacy-free way and authenticity of garbling will ensure a corrupt  $P_i$  does not get the certificate.  $\text{cert}_i$  follows the footsteps of  $\text{fair}_i$  with the following simplifications: (a) Input consistency need not be taken care across the executions implying that it is enough one garbler alone initiates a GC and the other garbler simply extends its support for verification. To divide the load fairly, we assign garbler  $P_j$  where  $i = (j + 1) \bmod 3$  to act as the generator of GC in  $\text{cert}_i$ . (b) The decoding information need not be committed or withheld. We use soft decoding that allows immediate decoding.

Similar to  $\text{fair}_i$ , at the end of the protocol, either  $P_i$  gets its certificate (either the key for 1 or the inputs themselves), or sets its flags (when GC and commitment do not match) or sets its corrupt set (when opening of encoded inputs fail).  $P_i$  outputs its certificate, the flag for the GC generator and corrupt set, to be used in the final protocol. The garblers output the key for 1, flag for its fellow garbler and the corrupt set. Notice that, when  $\text{cert}_i$  is composed in the bigger protocol,  $P_i$  will be in a position to identify the corrupt when the equality fails and the certificate is the inputs fed by the garblers. The protocol appears in Figure 2.

**Lemma 3.** *During  $\text{cert}_i$ ,  $P_\beta \notin \mathcal{C}_\alpha$  holds for honest  $P_\alpha, P_\beta$ .*

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

**Lemma 4.** *If an honest  $P_i$  has  $\mathcal{C}_i = \emptyset$  and  $\text{flag}_j = \text{flag}_k = 0$ , then,  $\text{cert}_i \neq \perp$ .*

*Proof.* The proof follows easily from the steps of the protocol.  $\square$

### 3.3 Protocol fair

Building on the intuition laid out before, we only discuss input consistency that is taken care in two steps: Inter-input consistency (across executions) and intra-input consistency (within an execution). In the former,  $P_i$ 's input as an evaluator in  $\text{fair}_i$  is tied with its input committed as garblers for its own garbled circuits in  $\text{fair}_j$  and  $\text{fair}_k$ . In the latter, the consistency of  $P_i$ 's input for both garbled circuits in  $\text{fair}_j$  (and similarly in  $\text{fair}_k$ ) is tackled. We discuss them one by one.

We tackle the former in a simple yet clever way without incurring any additional overhead. We explain the technique for enforcing  $P_1$ 's input consistency on input  $x_1$  as an evaluator during  $\text{fair}_1$  and as a garbler during  $\text{fair}_2, \text{fair}_3$  with respect to his GC  $\mathbf{C}_1$ . Since the protocol is symmetric in terms of the roles of the parties, similar tricks are adopted for  $P_2$  and  $P_3$ . Let in the first round of  $\text{fair}_1$ ,  $P_1$  shares its input  $x_1$  by handing  $x_{12}$  and  $x_{13}$  to  $P_2$  and  $P_3$  respectively. Now corresponding to  $\mathbf{C}_1$  during  $\text{fair}_2$ ,  $P_1$  and  $P_3$  who act as the garblers use  $x_{13}$  as the permutation vector  $p_{11}$  that defines the order of the commitments of the bits of  $x_1$ . Now input consistency of  $P_1$ 's input is guaranteed if  $m_{11}$  transferred by  $P_1$  in  $\text{fair}_2$  is same as  $x_{12}$ ,  $P_1$ 's share for  $P_2$  in  $\text{fair}_1$ . For an honest  $P_1$ , the above will be true since  $m_{11} = p_{11} \oplus x_1 = x_{13} \oplus x_1 = x_{12}$ . If the check fails, then  $P_2$  identifies  $P_1$  as corrupt. This simple check forces  $P_1$  to use the same input



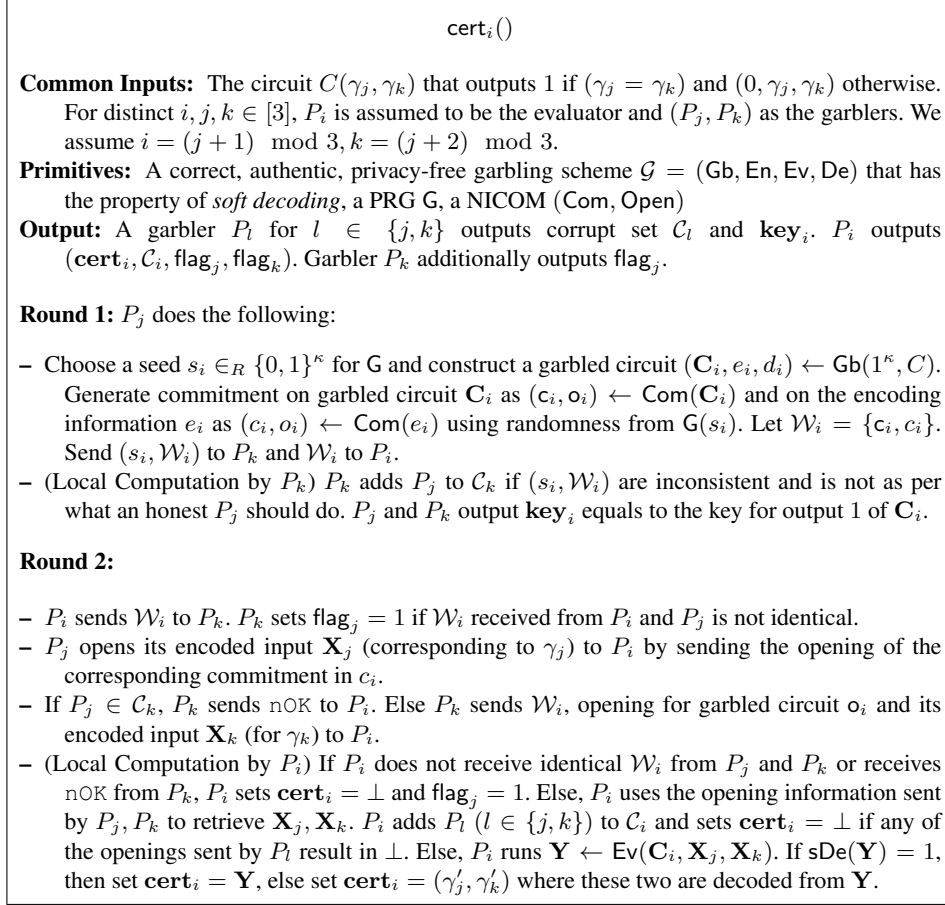


Fig. 2: Protocol  $\text{cert}_i$

in both  $\text{fair}_1$  and  $\text{fair}_2$  (corresponding to  $\mathbf{C}_1$ ). A similar trick is used to ensure input consistency of the input of  $P_1$  across  $\text{fair}_1$  and  $\text{fair}_3$  (corresponding to  $\mathbf{C}_1$ ) where  $P_1$  and  $P_2$  who act as the garblers use  $x_{12}$  as the permutation vector  $p_{11}$  for the commitments of the bits of  $x_1$ . The evaluator  $P_3$  in  $\text{fair}_3$  checks if  $m_{11}$  transferred by  $P_1$  in  $\text{fair}_3$  is same as  $x_{13}$  that  $P_3$  receives from  $P_1$  in  $\text{fair}_1$ . While the above technique enforces the consistency with respect to  $P_1$ 's GC, unfortunately, the same technique cannot be used to enforce  $P_1$ 's input consistency with respect to  $\mathbf{C}_2$  in  $\text{fair}_3$  (or  $\text{fair}_2$ ) since  $p_{21}$  cannot be set to  $x_{12}$  which is available to  $P_2$  only at the end of first round. While,  $P_2$  needs to prepare and broadcast the commitments to the encoding information in jumbled order as per permutation string  $p_{21}$  in the first round itself. We handle it differently as below.

The consistency of  $P_i$ 's input for both garbled circuits in  $\text{fair}_j$  (and similarly in  $\text{fair}_k$ ) is tackled via 'cheat-recovery mechanism' [Lin13]. We explain with respect to  $P_1$ 's input in  $\text{fair}_3$ .  $P_2$  prepares a ciphertext (cheat recovery box) with the input keys of  $P_1$  corresponding to the mismatched input bit in the two garbled circuits,  $\mathbf{C}_1$  and  $\mathbf{C}_2$  in

fair<sub>3</sub>. This ciphertext encrypts the the input shares of garblers that  $P_3$  misses, namely,  $x_{12}$  and  $x_{21}$ . This would allow  $P_3$  to compute the function on clear inputs directly. To ensure that the recovered missing shares are as distributed in fair<sub>1</sub> and fair<sub>2</sub>, the shares are not simply distributed but are committed via NICOM by the input owners and the openings are encrypted by the holders. Since there is no way for an evaluator to detect any mismatch in the inputs to and outputs from the two GCs as they are in encoded form, we use encryption scheme with special correctness (Definition 6) to enable the evaluator to identify the relevant decryptions. Crucially, we depart from the usual way of creating the cheat recovery boxes using conflicting encoded outputs. Based on whether the clear or encoded output comes out of honest  $P_3$  in round 3, corrupt garbler  $P_1$  feeding two different inputs to  $C_1$  and  $C_2$  can conclude whether its two different inputs lead to the same output or not, breaching privacy. Note that the decoding information cannot be given via this cheat recovery box that uses conflicting encoded outputs as key, as that would result in circularity.

Despite using the above fix, the mechanism as discussed above is susceptible to ‘selective failure attack’, an attack well-known in the 2-party domain. While in the latter domain, the attack is launched to breach the privacy of the evaluator’s input based on whether it aborts or not. Here, a corrupt garbler can prepare the ciphertexts in an incorrect way and can breach privacy of its honest co-garbler based on whether clear or encoded output comes out of the evaluator. We elaborate the attack in fair<sub>3</sub> considering a corrupt  $P_1$  and single bit inputs.  $P_1$  is supposed to prepare two ciphertexts corresponding to  $P_2$ ’s input bit using the following key combinations– (a) key for 0 in  $C_1$  and 1 in  $C_2$  and (b) vice-versa. Corrupt  $P_1$  may replace one of the ciphertexts using key based on encoded input 0 of  $P_2$  in both the GCs. In case  $P_2$  indeed has input 0 (that he would use consistently across the 2 GCs during fair<sub>3</sub>), then  $P_3$  would be able to decrypt the ciphertext and would send clear output in Round 3.  $P_1$  can readily conclude that  $P_2$ ’s input is 0. This attack is taken care via the usual technique of breaking each input bit to  $s$  number of xor-shares, referred as ‘XOR-tree approach’ [LP07] (probe-resistance matrix [LP07, SS13] can also be used; we avoid it for simplicity). The security is achieved except with probability  $2^{-(s-1)}$ .

Given that input consistency is enforced, at the end of round 2, apart from the three states– (a) no corruption and no conflict detected (b) corrupt identified (c) conflict detected, a party can be in yet another state. Namely, no corruption and no conflict detected and the party is able to open a ciphertext and compute  $f$  on clear. A corrupt party cannot be in this state since the honest parties would use consistent inputs and therefore the corrupt would not get access to conflicting encoded inputs that constitute the key of the ciphertexts. If any honest party is in this state, our protocol results in all parties outputting this output. In Round 3, this party can send the computed output along with the opening of the shares he recovered via the ciphertexts as ‘proof’ to convince the honest party of the validity of the output. The protocol fair appears in Figure 4 and the schematic diagram is given in Appendix D.1.

We now prove the correctness of fair.

**Lemma 5.** *During fair,  $P_j \notin C_i$  holds for honest  $P_i, P_j$ .*

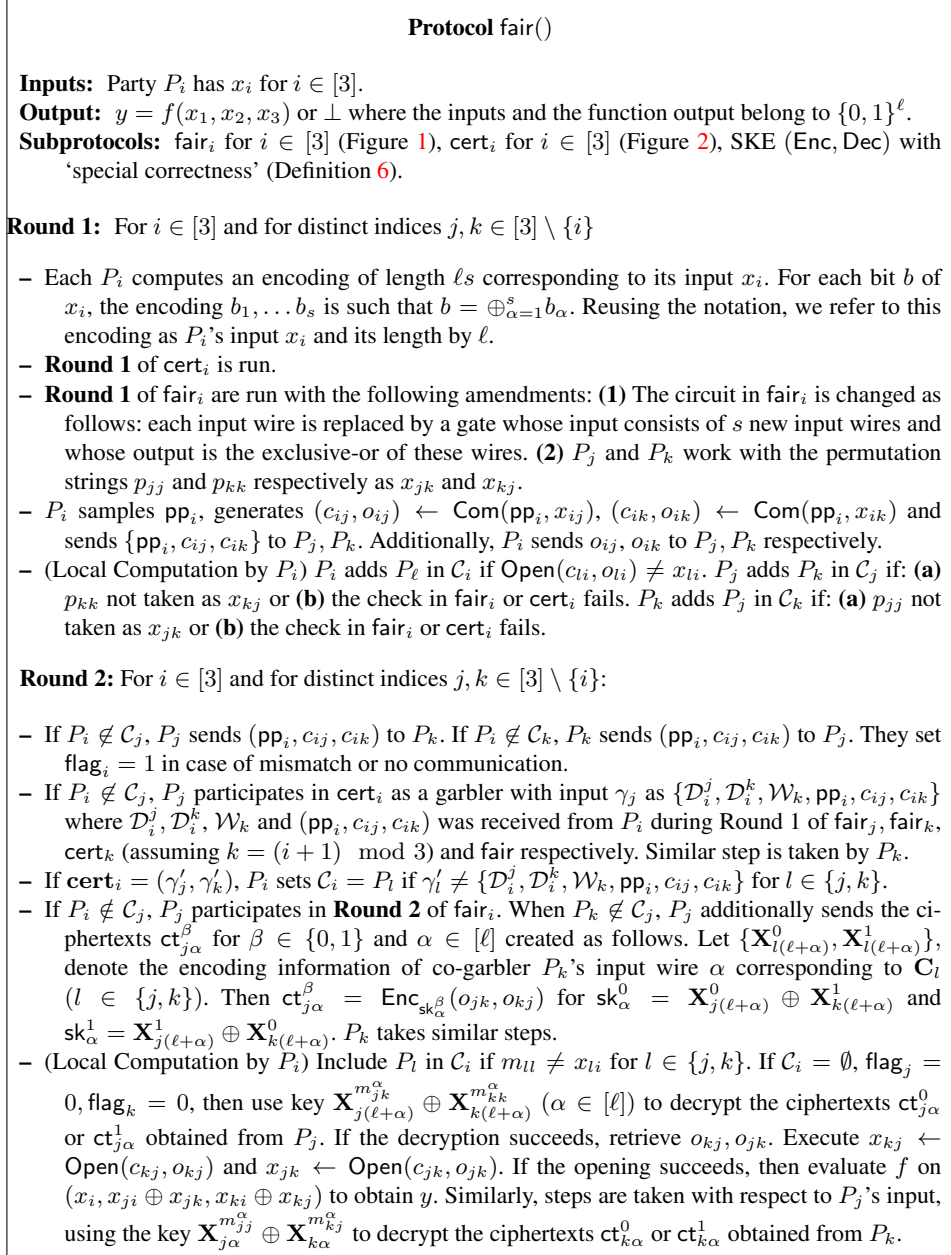


Fig. 3: A Three-Round Fair 3PC protocol

*Proof.* An honest  $P_i$  will not include  $P_j$  in its corrupt set in the sub-protocols  $\{\text{fair}_\alpha, \text{cert}_\alpha\}_{\alpha \in [3]}$  following Lemma 1, Lemma 3. Now we prove the statement individually investigating the three rounds of  $\text{fair}$ .

A party  $P_i$  is said to be in  $\text{st}_\alpha$  for  $\alpha \in [4]$  if the following conditions are satisfied. Let  $(\mathbf{Y}_i, C_j^{\text{dec}}, C_k^{\text{dec}})$ ,  $O_j^{\text{dec}}$  and  $O_k^{\text{dec}}$  denote the output of  $P_i$  in  $\text{fair}_i$ ,  $\text{fair}_j$  and  $\text{fair}_k$ , respectively. Let  $\text{cert}_i$ ,  $\text{key}_j$ , and  $\text{key}_k$  denotes the output of  $P_i$  in  $\text{cert}_i$ ,  $\text{cert}_j$  and  $\text{cert}_k$  respectively.

- (i)  $\text{st}_1$  (output is already computed): If  $y$  and proofs  $(o_{jk}, o_{kj})$  are computed in Round 2.
- (ii)  $\text{st}_2$  (no corruption and no conflict detected): If  $((C_i = \emptyset) \wedge (\text{flag}_j = 0) \wedge (\text{flag}_k = 0))$  (which implies  $\mathbf{Y}_i \neq \perp$  and  $\text{cert}_i \neq \perp$ )
- (iii)  $\text{st}_3$  (corruption detected): If  $(C_i \neq \emptyset)$
- (iv)  $\text{st}_4$  (conflict detected, but no corruption detected): If  $(\text{flag}_j = 1) \vee (\text{flag}_k = 1)$

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

- If in  $\text{st}_1$ , then send  $y$  to  $P_j, P_k$ . Send  $o_{jk}$  to  $P_j$  and  $o_{kj}$  to  $P_k$  as proofs.
- If in  $\text{st}_2$ , then send  $(\mathbf{Y}_i, \text{cert}_i, O_l^{\text{dec}})$  to  $P_l$  for  $l \in \{j, k\}$ .
- If in  $\text{st}_3$ , then send  $O_l^{\text{dec}}$  to  $P_l$  for  $l \in \{j, k\}$  only if  $P_l \notin C_i$ .
- If in  $\text{st}_4$ , then send  $z_l = \text{Enc}_{\text{key}_l}(O_l^{\text{dec}})$  to  $P_l$  only if  $\text{flag}_l = 1$ . If  $\text{flag}_j = 1$  and  $\text{cert}_j$  received from  $P_j$  is same as  $\text{key}_j$ , then set  $C_i = P_k$ . Similar steps are taken to check and identify if  $P_j$  is corrupt. Update state from  $\text{st}_4$  to  $\text{st}_3$  if corrupt is identified.
- If in  $\text{st}_1$ , then output  $y$ .
- If in  $\{\text{st}_2, \text{st}_3, \text{st}_4\}$  and if any other party is identified to be in  $\text{st}_1$ , namely if  $y$  is received from  $P_j$  or  $P_k$  with  $o_{ki}$  or  $o_{ji}$  respectively such that  $\text{Open}(\text{pp}_i, c_{li}, o_{li}) \neq \perp$  for  $l \in \{j, k\}$ , then output the received  $y$ .
- If in  $\text{st}_2$ , then compute  $y$  as follows: Retrieve  $O_i^{\text{dec}}$  from either  $z_i$  (with  $\text{cert}_i$  as the key) received from  $P_j$  or from direct communication of  $P_j$ . If  $d \leftarrow \text{eOpen}(\text{epk}_k, C_k^{\text{dec}}, O_i^{\text{dec}})$  is not  $\perp$ , then use  $d$  to compute  $y \leftarrow \text{De}(\mathbf{Y}_i^k, d)$ . Similar steps are executed with respect to  $P_k$ 's communication if  $y$  is not computed yet.
- If in  $\text{st}_3$ , then output  $y \leftarrow \text{De}(\mathbf{Y}_i^l, d)$  where  $\mathbf{Y}_l$  is received from (honest)  $P_l \notin C_i$  and decoding information  $d$  is known as garbler during  $\text{fair}_l$ . Otherwise output  $y = \perp$ .
- If in  $\text{st}_4$ , output  $y = \perp$ .

Fig. 4: A Three-Round Fair 3PC protocol

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

**Lemma 6.** *No corrupt party can be in  $\text{st}_1$  by the end of Round 1, except with negligible probability.*

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

**Definition 1.** A party  $P_i$  is said to be ‘committed’ to a unique input  $x_i$ , if  $P_j$  holds  $(c_{ij}, c_{ik}, o_{ij}, x_{ij})$  and  $P_k$  holds  $(c_{ij}, c_{ik}, o_{ik}, x_{ik})$  such that: **(a)**  $x_i = x_{ij} \oplus x_{ik}$  and **(b)**  $c_{ij}$  opens to  $x_{ij}$  via  $o_{ij}$  and likewise,  $c_{ik}$  opens to  $x_{ik}$  via  $o_{ik}$ .

We next prove that a corrupt party must have committed its input if some honest party is in  $st_1$  or  $st_2$ . To prove correctness, the next few lemmas then show that an honest party computes its output based on its own output or encoded output if it is in  $st_1$  or  $st_2$  or relies on the output or encoded output of the other *honest* party. In all cases, the output will correspond to the committed input of the corrupt party.

**Lemma 7.** *If an honest party is in  $\{st_1, st_2\}$ , then corrupt party must have committed a unique input.*

*Proof.* An honest  $P_i$  is in  $\{st_1, st_2\}$  only when  $C_i = \emptyset$ ,  $flag_j = 0$ ,  $flag_k = 0$  hold at the end of Round 2. Assume  $P_k$  is corrupt.  $P_k$  has not committed to a unique  $x_k$  implies either it has distributed different copies of commitments  $(c_{ki}, c_{kj})$  to the honest parties or distributed incorrect opening information to some honest party. In the former case,  $flag_k$  will be set by  $P_i$ . In the latter case, at least one honest party will identify  $P_k$  to be corrupt by the end of Round 1. If it is  $P_i$ , then  $C_i \neq \emptyset$ . Otherwise,  $P_j$  populates its corrupt set with  $P_k$ , leading to  $P_i$  setting  $flag_k = 1$  in Round 2.  $\square$

**Lemma 8.** *If an honest party is in  $st_1$ , then its output  $y$  corresponds to the unique input committed by the corrupt party.*

*Proof.* An honest  $P_i$  is in  $st_1$  only when  $C_i = \emptyset$ ,  $flag_j = 0$ ,  $flag_k = 0$  hold at the end of Round 2 and it computes  $y$  via decryption of the ciphertexts  $ct$  sent by either  $P_j$  or  $P_k$ . Assume  $P_k$  is corrupt. By Lemma 7,  $P_k$  has committed to its input. The condition  $flag_j = 0$  implies that  $P_k$  exchanges the commitments on the shares of  $P_j$ ’s input, namely  $\{c_{ji}, c_{jk}\}$ , honestly. Now if  $P_i$  opens honest  $P_j$ ’s ciphertext, then it unlocks the opening information for the missing shares, namely  $(o_{kj}, o_{jk})$  corresponding to common and agreed commitments  $(c_{kj}, c_{jk})$ . Using these it opens the missing shares  $x_{kj} \leftarrow \text{Open}(c_{kj}, o_{kj})$  and  $x_{jk} \leftarrow \text{Open}(c_{jk}, o_{jk})$  and finally computes output on  $(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ . Next, we consider the case when  $P_i$  computes  $y$  by decrypting a  $ct$  sent by corrupt  $P_k$ . In this case, no matter how the ciphertext is created, the binding property of NICOM implies that  $P_k$  will not be able to open  $c_{jk}, c_{kj}$  to anything other than  $x_{jk}, x_{kj}$  except with negligible probability. Thus, the output computed is still as above and the claim holds.  $\square$

**Lemma 9.** *If an honest party is in  $st_2$ , then its encoded output  $Y$  corresponds to the unique input committed by the corrupt party.*

*Proof.* An honest  $P_i$  is in  $\text{st}_2$  only when  $\mathcal{C}_i = \emptyset$ ,  $\text{flag}_j = 0$ ,  $\text{flag}_k = 0$  hold at the end of Round 2. The conditions also imply that  $P_i$  has computed  $\mathbf{Y}_i$  successfully (due to Lemma 2) and  $P_k$  has committed to its input (due to Lemma 7). Now we show that  $\mathbf{Y}_i$  correspond to the unique input committed by the corrupt  $P_k$ . We first note that  $P_k$  must have used the same input for both the circuits  $\mathbf{C}_j$  and  $\mathbf{C}_k$  in  $\text{fair}_i$ . Otherwise one of the ciphertexts prepared by honest  $P_j$  must have been opened and  $y$  would be computed, implying  $P_i$  belongs to  $\text{st}_1$  and not in  $\text{st}_2$  as assumed. We are now left to show that the input of  $P_k$  for its circuit  $\mathbf{C}_k$  in  $\text{fair}_i$  is the same as the one committed.

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

**Lemma 10.** *If an honest party is in  $\text{st}_2$ , then its output  $y$  corresponds to the unique input committed by the corrupt party.*

*Proof.* Note that an honest party  $P_i$  in  $\text{st}_2$  either uses  $y$  of another party in  $\text{st}_1$  or computes output from its encoded output  $\mathbf{Y}_i$ . The proof for the former case goes as follows. By Lemma 6, a corrupt  $P_k$  can never be in  $\text{st}_1$ . The correctness of  $y$  computed by an honest  $P_j$  follows directly from Lemma 8. For the latter case, Lemma 9 implies that  $\mathbf{Y}_i$  corresponds to the unique input committed by the corrupt party. All that needs to be ensured is that  $P_i$  gets the correct decoding information. The condition  $\text{flag}_j = \text{flag}_k = 0$  implies that the commitment to the decoding information is computed and distributed correctly for both  $\mathbf{C}_j$  and  $\mathbf{C}_k$ . Now the binding property of eNICOM ensures that the decoding information received from either  $P_j$  (for  $\mathbf{C}_k$ ) or  $P_k$  (for  $\mathbf{C}_j$ ) must be correct implying correctness of  $y$  (by correctness of the garbling scheme).  $\square$

**Lemma 11.** *If an honest party is in  $\text{st}_3$  or  $\text{st}_4$ , then its output  $y$  corresponds to the unique input committed by the corrupt party.*

*Proof.* An honest party  $P_i$  in  $\text{st}_3$  either uses  $y$  of another party in  $\text{st}_1$  or computes output from encoded output  $\mathbf{Y}_j$  of  $P_j$  who it identifies as honest. For the latter case note that an honest  $P_j$  will never be identified as corrupt by  $P_i$ , due to Lemma 5. The claim now follows from Lemma 6, Lemma 8 and the fact that corrupt  $P_k$  cannot forge the ‘proof’  $o_{ij}$  (binding of NICOM) for the former case and from Lemma 9 and the fact that it possesses correct decoding information as a garbler for  $\mathbf{Y}_j$  for the latter case. An honest party  $P_i$  in  $\text{st}_4$  only uses  $y$  of another party in  $\text{st}_1$ . The lemma follows in this case via the same argument as before.  $\square$

**Theorem 1.** *Protocol fair is correct.*

*Proof.* In order to prove the theorem, we show that if an honest party, say  $P_i$  outputs  $y$  that is not  $\perp$ , then it corresponds to  $x_1, x_2, x_3$  where  $x_j$  is the input committed by  $P_j$  (Definition 1). We note that an honest  $P_i$  belong to one among  $\{\text{st}_1, \text{st}_2, \text{st}_3, \text{st}_4\}$  at the time of output computation. The proof now follows from Lemmas 7,8,10,11.  $\square$

Fairness implies: (a) if a corrupt party gets the output then so does the honest parties; (b) if an honest party gets the output then so does the other parties. We give the intuition for both below starting with (a). The formal proof appears in Appendix D.2.

A corrupt  $P_k$  cannot be in  $\text{st}_1$  (due to Lemma 6). The only way it can retrieve the output is by having an honest party in  $\text{st}_1$  or  $\text{st}_2$ . An honest party in  $\text{st}_3$  only releases the decoding information and it never release it to a corrupt party (Lemma 5 implies it identifies the honest party correctly). An honest party in  $\text{st}_4$  releases the encrypted decoding information  $z_k$  under key  $\text{key}_k$  to  $P_k$  conditionally when  $\text{flag}_k = 1$ . The condition  $\text{flag}_k = 1$  implies that  $P_k$  must have distributed the common information incorrectly and so  $\gamma_i$  and  $\gamma_j$  are not same. This further implies  $\text{cert}_k$  is not same as  $\text{key}_k$  and so  $P_k$  does not have access to the key to open  $z_k$  and cannot recover the decoding information. So the corrupt  $P_k$  getting the output implies that at least one honest party is in  $\{\text{st}_1, \text{st}_2\}$ . Lemma 7 implies that in this case,  $P_k$  must have committed to a unique input. By Lemma 8 and Lemma 10, the  $y$  and encoded output  $\mathbf{Y}$  computed by any honest party in  $\text{st}_1$  and in  $\text{st}_2$  respectively will correspond  $P_k$ 's committed input. Further, if  $P_k$  computes encoded output  $\mathbf{Y}_k$ , it also correspond to  $P_k$ 's committed input. So no matter how the corrupt party compute the output, it will be with respect to unique  $(x_1, x_2, x_3)$ . We need to show that both honest parties receive the same output. This easily follows when at least one honest party is in  $\text{st}_1$ . We now prove the lemma based on the following cases. (a) Both  $P_i, P_j$  are in  $\text{st}_2$ : They receive the decoding information from each other on the clear and use their respective computed encoded output to compute the output  $y$ . (b)  $P_i$  is in  $\text{st}_2$  and  $P_j$  in  $\text{st}_3$ :  $P_i$  uses the decoding information sent exclusively to him by  $P_j$  and decode the output as in the previous case.  $P_j$  uses the encoded output of  $P_i$ ,  $\mathbf{Y}_i$  and its decoding information (held as a garbler) to compute the output. (c)  $P_i$  is in  $\text{st}_2$  and  $P_j$  in  $\text{st}_4$ :  $P_j$  must be in  $\text{st}_4$  because of  $\text{flag}_j = 1$ . If  $\text{flag}_k = 1$ ,  $P_i$  will have the same status for this flag and would belong to  $\text{st}_4$ . Now since  $\text{flag}_i = 1$ ,  $P_j$  sends encryption of the decoding information  $z_i$  to  $P_i$  who can use  $\text{cert}_i$  to decrypt  $z_i$  and compute the output as in the previous two cases.  $P_j$ , on noting that  $\text{flag}_i = 1$ , yet  $P_i$  obtained  $\text{cert}_i = \text{key}_i$ , will identify  $P_k$  to be corrupt, upgrade to  $\text{st}_3$  and compute the output as in the previous case.

Next, we argue for part (b). For an honest party to compute the output  $y$ , at least one honest party must be in  $\{\text{st}_1, \text{st}_2\}$ . If both belong to  $\{\text{st}_3, \text{st}_4\}$ , then neither  $P_k$  has committed any input (due to Lemma 7) nor anyone gets the output. The latter follows by the argument below. An honest party in  $\text{st}_3$  only outputs based on the encoded output of the other honest party. But since the other honest party is in  $\{\text{st}_3, \text{st}_4\}$ , it will output  $\perp$ . An honest party in  $\text{st}_4$  outputs  $\perp$ , except for the case it finds one in  $\text{st}_1$  which is not true for both  $P_j$  and  $P_k$  (Lemma 6). The corrupt  $P_k$  does not get the output too following the fact that it cannot be in  $\text{st}_1$  (Lemma 6) and it does not receive decoding information from an honest party. An honest party  $P_i$  in  $\text{st}_3$  sends the decoding information only to the *identified* honest party. An honest party  $P_i$  in  $\text{st}_4$  may send the encrypted decoding information  $z_k$  under key  $\text{key}_k$  to  $P_k$  when  $\text{flag}_k = 1$ . But the condition  $\text{flag}_k = 1$  implies that  $P_k$  must have distributed the common information incorrectly and so  $\gamma_i$  and  $\gamma_j$  are not same. This further implies  $\text{cert}_k$  is not same as  $\text{key}_k$  and so  $P_k$  does not have access to the key to open  $z_k$  and cannot recover the opening information. Now

we are left to show that when at least one honest party is in  $\{st_1, st_2\}$ , then everyone gets the output. This already follows from the argument given for the other direction.

## 4 2-round 3PC with Unanimous Abort

This section presents a tight upper bound for 3PC achieving unanimous abort in the setting with pair-wise private channels and a broadcast channel. The impossibility of one-round protocol in the same setting follows from “residual function” attack [HLP11]. Our result from Section 6.2 rules out the possibility of achieving unanimous abort in the absence of a broadcast channel in two rounds. This protocol can be used to yield a round-optimal fair protocol with broadcast (lower bound in Section 6.1) by application of the transformation of [IKP<sup>+</sup>16] that compiles a protocol with unanimous abort to a fair protocol via evaluating the circuits that compute shares (using error-correcting secret sharing) of the function output using the protocol with unanimous abort and then uses an additional round for reconstruction of the output.

In an attempt to build a protocol with unanimous abort, we note that any protocol with unanimous abort must be robust to any potential misbehaviour launched via the private communication in the second round. Simply because, there is no way to report the abort to the other honest party who may have seen honest behaviour from the corrupt party all along and has got the output, leading to selective abort. Our construction achieves unanimity by leveraging the availability of the broadcast channel to abort when a corrupt behaviour is identified either in the first round or in the broadcast communication in the second round, and behaving robustly otherwise. In summary, if the corrupt party does not strike in the first round and in the broadcast communication of the second round, then our construction achieves robustness.

Turning to the garbled circuit based constructions such as the two-round protocol of [IKKP15] achieving selective abort or the composition of three copies of the sub-protocol  $fair_i$  of  $fair$ , we note that the second round private communication that involves encoding information for inputs is crucial for computing the output and cannot transit via broadcast because of input privacy breach. A bit elaborately, the transfer of the encoding information for the inputs of the garblers can be completed in the first round itself and any inconsistency can be handled via unanimous abort in the second round. However, a similar treatment for the encoding information of the shares of the evaluator seems impossible as they are transferred to garblers only in the first round. We get past this seemingly impossible task via a clever ‘two-part release mechanism’ for the encoding information of the shares of the evaluator. Details follow.

Similar to protocol  $fair$ , we build our protocol  $ua$  upon three parallel executions of a sub-protocol  $ua_i$  ( $i \in [3]$ ), each comprising of two rounds and with each party  $P_i$  enacting the role of the evaluator once. With  $fair_i$  as the starting point, each sub-protocol  $ua_i$  allows the parties to reach agreement on whether the run was successful and the evaluator got the output or not. A flag  $flag_i$  is used as an indicator. The protocol  $ua$  then decides on unanimous abort if at least one of the flags from the three executions  $ua_i$  for  $i \in [3]$  is set to true. Otherwise, the parties must have got the output. Input consistency checks ensure that the outputs are identical. Intra-execution input consistency is taken care by cheat-recovery mechanism (similar and simplified version of what protocol  $fair$



uses), while inter-execution input consistency is taken care by the same trick that we use in our fair protocol. Now looking inside  $ua_i$ , the challenge goes back to finding a mechanism for the honest evaluator to get the output when a corrupt party behaves honestly in the first round and in the broadcast communication of the second round. In other words, its private communication in the second round should not impact robustness. This is where the ‘two-part release mechanism’ for the encoding information of the shares of the evaluator kicks in. It is realized by tweaking the function to be evaluated as  $f(x_j, x_k, (z_j \oplus r_j) \oplus (z_k \oplus r_k))$  in the instance  $ua_i$  where  $P_i$  enacts the role of the evaluator. Here  $r_j, r_k$  denote random pads chosen by the garblers  $P_j, P_k$  respectively in the first round. The encoding information for these are released to  $P_i$  *privately* in the first round itself. Any inconsistent behaviour in the first round is detected, the flag is set and the protocol exits with  $\perp$  unanimously. Next,  $z_j$  and  $z_k$  are the offsets of these random pads with the actual shares of  $P_i$ ’s input and are available only at the end of first round. The encoding information for these offsets and these offsets themselves are transferred via broadcast in the second round for public verification. As long as the pads are privately communicated, the offsets do not affect privacy of the shares of  $P_i$ ’s input. Lastly, note that the encoding information for a garbler’s input for its own generated circuit can be transferred in the first round itself. This ensures that a corrupt garbler misbehaves either in the first round or in the broadcast communication in the second round or lets the evaluator get the output via its own GC. We describe execution  $ua_i$ , assuming input consistency. Entwining the three executions, tackling the input consistency and the final presentation of protocol  $ua$  are done next. Lastly, we present the security proof.

#### 4.1 Protocol $ua_i$

With the goal to achieve agreement among the honest parties regarding whether the evaluator got the output or not,  $ua_i$  starts with  $fair_i$  and makes the following changes. First, the broadcast channel is used to reach agreement on the commitments to GCs and the encoding information. Second, a garbling scheme with soft decoding property is used to allow immediate output decoding. Third, a garbler opens its encoded input for its own GC in the first round itself. In addition, we implement the two-part release mechanism for  $P_i$ ’s shares where apart from the garblers,  $P_i$  too broadcasts the offsets in the second round. A flag  $flag_i$  is used to keep track if a complaint is raised for the first round communication by broadcast in the second round or the offsets broadcasted in parallel by both  $P_i$  and respective garblers do not match or the opening of the encoded input for the offsets fails. When  $flag_i$  remains to be false for the honest parties, an honest  $P_i$  must be able to evaluate and output from the GC prepared by the corrupt garbler. Because, the commitments to that GC and encoding information has been scrutinized by the honest co-garbler, the encoded input of the corrupt party has been verified by the evaluator, the release of the encoded inputs for the shares of the evaluator has been verified publicly and the offsets themselves matched. Lastly, since the flag when set to be true by any honest party in the end of first round can be propagated to all in the second round and is only set based on the broadcasts in the second round, all honest parties exit  $ua_i$  with an agreement on  $flag_i$ . We now present our protocol in Figures 5-6 assuming input consistency and prove its properties needed later.

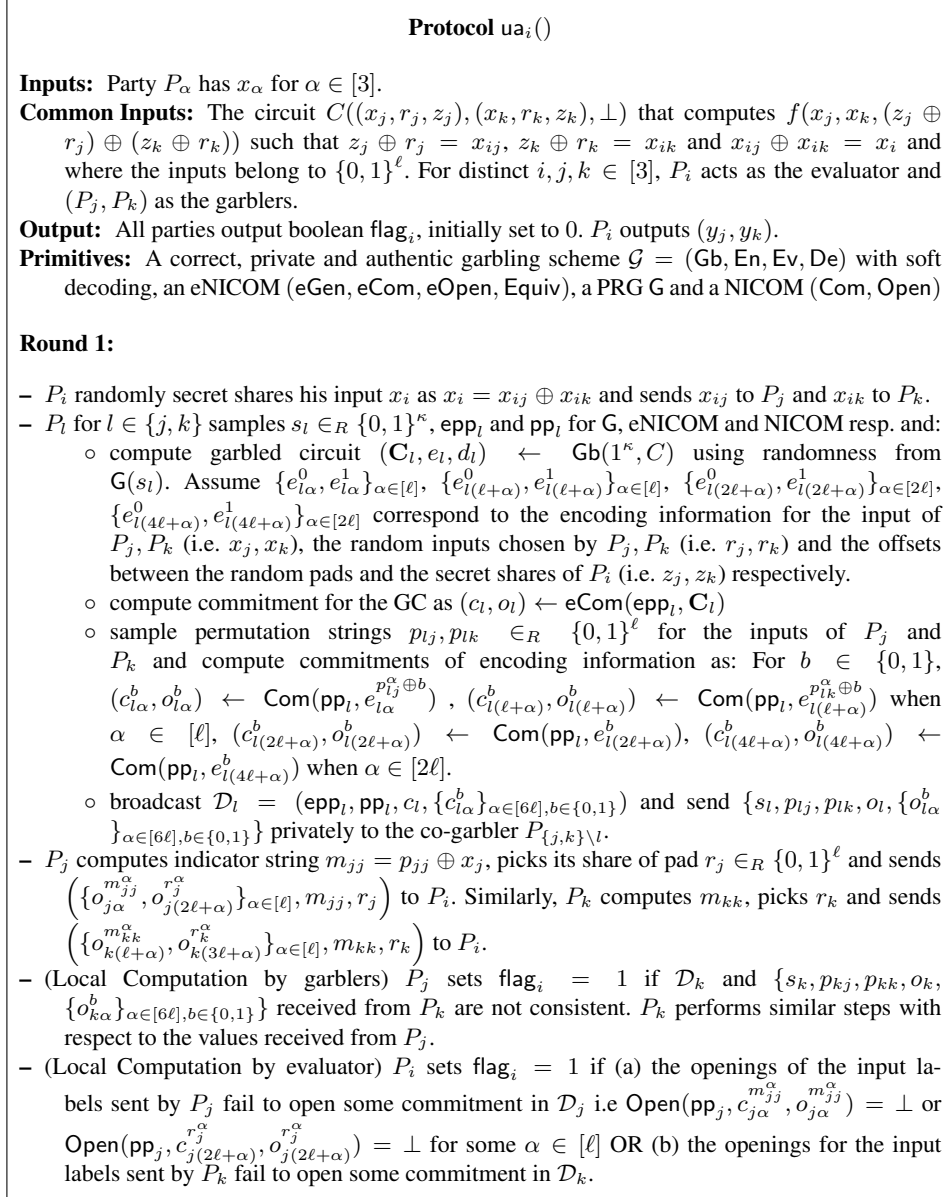


Fig. 5: Protocol  $ua_i$

**Lemma 12.** *At the end of protocol  $ua_i$ , all honest parties output the same flag $_i$ .*

*Proof.* We have two cases based on whether atleast one honest party set  $\text{flag}_i = 1$  at the end of Round 1. If this is true, then the honest party would broadcast abort in Round 2 and all honest parties would output  $\text{flag}_i = 1$ . Otherwise, an honest party sets  $\text{flag}_i$  based on the following conditions (a) abort was broadcast in Round 2 or (b) either

**Contd. Protocol  $ua_i()$**

**Round 2:**

- $P_j$  broadcasts **abort** if  $\text{flag}_i = 1$ . Else, it computes its indicator string  $m_{kj} = p_{kj} \oplus x_j$  for  $P_k$ 's circuit and the offset  $z_j = x_{ij} \oplus r_j$ , sends  $(\text{OK}, o_k, \{o_{k\alpha}^{m_{kj}}, o_{k(2\ell+\alpha)}^{r_j^\alpha}, o_{k(4\ell+\alpha)}^{z_j^\alpha}\}_{\alpha \in [\ell]}, m_{kj})$  privately to  $P_i$  and broadcasts  $\mathcal{W}_j = (z_j, \{o_{j(4\ell+\alpha)}^{z_j^\alpha}\}_{\alpha \in [\ell]})$ .  $P_k$  performs similar steps.
  - $P_i$  broadcasts **abort** if  $\text{flag}_i = 1$ . Else, it broadcasts  $z_j = x_{ij} \oplus r_j$  and  $z_k = x_{ik} \oplus r_k$
  - Every party sets  $\text{flag}_i = 1$  if **(a)** **abort** was received or sent via broadcast in Round 2 OR **(b)** either  $z_j$  broadcast by  $(P_j, P_i)$  or  $z_k$  broadcast by  $(P_k, P_i)$  do not match OR **(c)**  $\mathcal{D}_j, \mathcal{W}_j$  is not consistent i.e  $\text{Open}(\text{pp}_j, c_{j(4\ell+\alpha)}^{z_j^\alpha}, o_{j(4\ell+\alpha)}^{z_j^\alpha}) = \perp$  or similarly  $\mathcal{D}_k, \mathcal{W}_k$  is not consistent.
  - (Local Computation by  $P_i$ ) Output  $y_j = y_k = \perp$  if  $\text{flag}_i = 1$ . Else, with respect to  $\mathbf{C}_j$ :
    - o open  $\mathbf{C}_j \leftarrow \text{eOpen}(\text{pp}_j, c_j, o_j)$  where the opening is received from  $P_k$ .
    - o open  $\mathbf{X}_j^\alpha = \text{Open}(\text{pp}_j, c_{j\alpha}^{m_{jj}^\alpha}, o_{j\alpha}^{m_{jj}^\alpha}), \mathbf{R}_j^\alpha = \text{Open}(\text{pp}_j, c_{j(2\ell+\alpha)}^{r_j^\alpha}, o_{j(2\ell+\alpha)}^{r_j^\alpha}),$  and  $\mathbf{Z}_j^\alpha = \text{Open}(\text{pp}_j, c_{j(4\ell+\alpha)}^{z_j^\alpha}, o_{j(4\ell+\alpha)}^{z_j^\alpha}),$  for the openings received from  $P_j$ .
    - o open  $\mathbf{X}_k^\alpha = \text{Open}(\text{pp}_j, c_{j(\ell+\alpha)}^{m_{jk}^\alpha}, o_{j(\ell+\alpha)}^{m_{jk}^\alpha}), \mathbf{R}_k^\alpha = \text{Open}(\text{pp}_j, c_{j(3\ell+\alpha)}^{r_k^\alpha}, o_{j(3\ell+\alpha)}^{r_k^\alpha})$  and  $\mathbf{Z}_k^\alpha = \text{Open}(\text{pp}_j, c_{j(5\ell+\alpha)}^{z_k^\alpha}, o_{j(5\ell+\alpha)}^{z_k^\alpha})$  for  $\alpha \in [\ell]$ , for openings are received from  $P_k$ .
    - o If any of the above openings fail, set  $y_j = \perp$ . Else set  $\mathbf{X} = \mathbf{X}_j | \mathbf{X}_k | \mathbf{R}_j | \mathbf{R}_k | \mathbf{Z}_j | \mathbf{Z}_k$ , run  $\mathbf{Y}_j \leftarrow \text{Ev}(\mathbf{C}_j, \mathbf{X})$  and  $y_j \leftarrow \text{sDe}(\mathbf{Y}_j)$ .
- Similar steps as above with respect to  $\mathbf{C}_k$  is executed to compute  $\mathbf{Y}_k$  and  $y_k$ .

Fig. 6: Protocol  $ua_i$

$z_j$  broadcast by  $(P_j, P_i)$  or  $z_k$  broadcast by  $(P_k, P_i)$  do not match or (c)  $(\mathcal{D}_j, \mathcal{W}_j)$  or  $(\mathcal{D}_k, \mathcal{W}_k)$  is inconsistent. All these checks are with respect to broadcast messages. Therefore, we can conclude that every honest party will output identical  $\text{flag}_i$ .  $\square$

**Lemma 13.** *Assuming input consistency, if  $\text{flag}_i = 0$ , then  $y_k \neq \perp$  where  $P_k$  is corrupt.*

*Proof.* First, Lemma 12 implies that both  $P_i, P_j$  output identical  $\text{flag}_i = 0$ . Now  $\text{flag}_i = 0$  implies that: **(a)**  $\mathbf{C}_k$  and the commitments to the encoding information are computed correctly; **(b)** the opening of encoding information  $\mathbf{X}_k, \mathbf{R}_k$  for  $\mathbf{C}_k$  is correct in Round 1 with high probability due to binding property of eNICOM and NICOM; **(c)** the opening of the remaining encoding information  $\mathbf{Z}_k$  is correct with high probability due to binding property of NICOM.  $P_j$  being honest would open the encoding relevant to his input for  $\mathbf{C}_k$ , namely,  $\mathbf{X}_j, \mathbf{R}_j, \mathbf{Z}_j$ . So  $P_i$  has got complete encoded input  $\mathbf{X}$  for  $\mathbf{C}_k$  and will evaluate  $\mathbf{C}_k$  to obtain  $y_k$ . Thus, if  $\text{flag}_i = 0$ , then  $y_k$  will not be  $\perp$ .  $\square$

## 4.2 Protocol $ua$

Our two-round 3PC protocol  $ua$  achieving unanimous abort composes  $ua_i$  for  $i \in [3]$  in parallel. Assuming input consistency, entwining the three executions requires tapping all the flags returned by the three executions and outputting the result computed as an evaluator when none of them are set to true and  $\perp$ , otherwise. This works since when a

flag for an execution  $ua_i$  is false, then the evaluator  $P_i$  is guaranteed to get the output. The challenge that remains to handle is input consistency within and across executions which ensures the outputs computed are the same irrespective of the execution and GC. The inter-execution input consistency, i.e the consistency of the input committed by  $P_i$  in  $ua_i$  and the inputs given to the GCs constructed by  $P_i$  as garbler in the remaining two executions are enforced using the same trick that we use in fair via setting the permutation strings as the shares of the parties' input.

Dealing with the input consistency within an execution  $ua_i$  to make sure the garblers provide the same input for both the GCs without inflating the round complexity constitutes yet another challenge. Noting that this misbehaviour has no way to show up in the common flag as this is targeted via the private communication in the second round, the evaluator must find a way to robustly compute the output when conflicted outputs are computed from the two garbled circuits. This output must be based on the input of the corrupt garbler that it has committed as an evaluator and received output based on. We use the trick of "proof-of-cheating" mechanism [Lin13] to enable an (honest) evaluator with conflicting outputs to retrieve the inputs committed by both garblers in their respective instances. To be specific, the output keys corresponding to the mismatched output bit in the two garbled circuits, say  $C_1$  and  $C_3$  in  $ua_2$ , enables the evaluator  $P_2$  to unlock the missing shares, namely,  $x_{31}$  and  $x_{13}$  of the two garblers from  $ua_3$  and  $ua_1$  respectively. To ensure that the recovered missing shares are as distributed in  $ua_1$  and  $ua_3$ , the shares are committed via NICOM by the input owners and the openings are encrypted by the holders (as in fair). The binding of NICOM, prevents a corrupt  $P_1$  to lie on  $(x_{13}, x_{31})$ . This allows the honest party to compute the same output that  $P_1$  gets from  $ua_1$ . Lastly, the flag in execution  $ua_i$  also takes into account consistent dealing of the commitments by its evaluator  $P_i$ . Our protocol appears in Figure 7, the proof of correctness and the proof of security below. We use Definition 1 for input commitment.

**Lemma 14.** *If a corrupt party  $P_k$  has not committed its input or does not use the committed input in its GCs in  $\{ua_i, ua_j\}$ , then each honest party outputs  $y = \perp$ .*

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

**Theorem 2.** *Protocol  $ua$  is correct.*

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

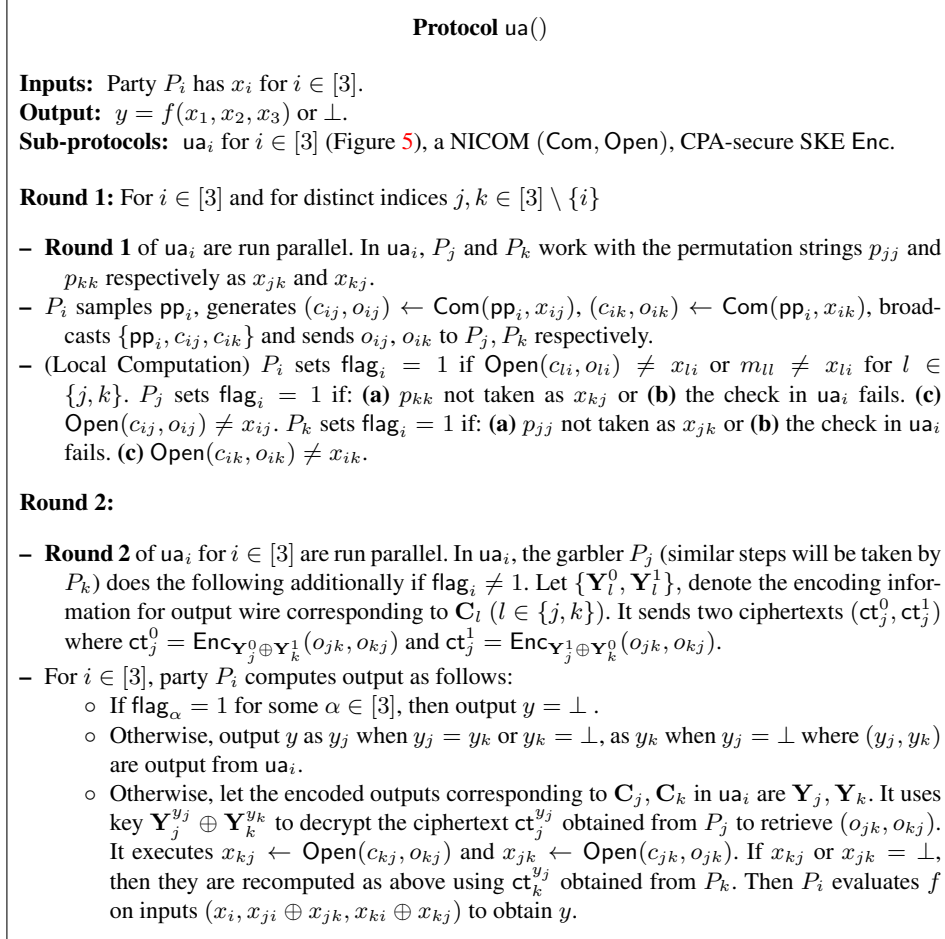


Fig. 7: A Two-Round 3PC protocol achieving unanimous abort

- $y = y_k$ . Follows from Lemma 13, 14.
- $y \neq y_k$ . In this case,  $y \neq y_j$  either as  $y$  is set to  $y_j$  when  $y_j = y_k$  or  $y_k = \perp$ . Following Lemma 13,  $y_k$  cannot be  $\perp$ . So it must be that  $P_i$  retrieves the output via opening the ciphertexts. If the output is computed just from the ciphertext of honest  $P_j$ , then  $y$  is computed as  $f(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$  using openings  $o_{kj}, o_{jk}$  given by  $P_j$ . Since an honest  $P_j$  correctly reveals the opening  $o_{kj}$  of the share of  $P_k$ 's input given to  $P_j$  and  $o_{jk}$  corresponding to his input share,  $f(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$  corresponds to the correct value. If the output is computed from the ciphertext of corrupt  $P_k$ , then  $y$  computed must be still as above as a corrupt  $P_k$  cannot open the shares  $x_{jk}, x_{kj}$  in an incorrect way (following binding property of NICOM).

□

The intuition for achieving unanimous abort follows from the correctness and Lemma 12 that implies the honest parties will be on the same page for all flags. The formal proof appears in Appendix E.

## 5 3-round 3PC with Guaranteed Output Delivery

In this section, we present a three-round 3PC protocol, given access to pairwise-private channels and a broadcast channel. The protocol is round-optimal following 3-round lower bound for fair 3PC proven in Section 6.1. The necessity of the broadcast channel for achieving guaranteed output delivery with strict honest majority follows from [CHOR16].

Our tryst starts with the known generic transformations that are relevant such as the transformations from the unanimous abort to (identifiable) fair protocol [IKP<sup>+</sup>16] or identifiable fair to guaranteed output delivery [CL14]. However, these transformations being non-round-preserving do not turn out to be useful. Turning a 2-round protocol offering unanimous (or even selective) abort with identifiability (when the honest parties learn about the identity of the corrupt when deprived of the output) to a 3-round protocol with guaranteed output delivery in a black-box way show some promise. The third round can be leveraged by the honest parties to exchange their inputs and compute output on the clear. We face two obstacles with this approach. First, there is neither any known 2-round construction for selective / unanimous abort with identifiability nor do we see how to transform our unanimous abort protocol to one with identifiability in two rounds. Second, when none of the parties (including the corrupt) receive output from the selective / unanimous abort protocol and the honest parties compute it on the clear in the third round by exchanging their inputs and taking a default value for the input of the corrupt party, it is not clear how the corrupt party can obtain the same output (note that the ideal functionality demands delivering the output to the adversary).

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

the output using a default input of the corrupt party by the end of Round 2 and hands it over to others in Round 3.

In the following, we describe one execution  $\text{god}_i$ . Entwining the three executions, tackling the input consistency and the final presentation of protocol  $\text{god}$  are done next. The security proof appears in Appendix F.

### 5.1 Protocol $\text{god}_i$

Recall that the goal of  $\text{god}_i$  for  $i \in [3]$  comprising of two rounds, is either successful computation of output or successful identification of the corrupt party by the evaluator  $P_i$ . Starting with the ideas of  $u_{a_i}$ , we note that  $u_{a_i}$  only ensures detection of the corrupt party by some honest party that is not necessarily the evaluator in case of a failed output computation. Specifically, a garbler would identify his co-garbler to be corrupt when the broadcast communication of co-garbler is not consistent with the privately shared randomness. In such a case, the evaluator neither gets the output nor has any clue on the identity of the corrupt, which is not in accordance with the goal of  $\text{god}_i$ . In the absence of broadcast,  $\text{fair}_i$  gives even weaker guarantee where the best any party gets to know is a conflict. The above is handled by having the garblers send their inputs on clear to the evaluator on finding inconsistent behaviour of the fellow garbler in the first round. If both the garblers are in conflict with each other, the evaluator gets their inputs and computes the function on clear. Otherwise, the evaluator can either evaluate at least one of the GCs or identify the corrupt. Lastly, as we do not require unanimity of any form at the end of two rounds, we simplify  $\text{god}_i$  by removing the two-part release mechanism and the flag altogether. Like  $u_{a_i}$ , we do not take care of the possibility of a corrupt garbler handing out inconsistent input for the two GCs in  $\text{god}_i$ . This is taken care in the main protocol  $\text{god}$  via the input consistency.  $P_i$  outputs  $(y = (y_j, y_k), \mathbf{Y}_i = (\mathbf{Y}_i^j, \mathbf{Y}_i^k), \mathcal{C}_i)$ , the outputs computed from two GCs, the encoded outputs and its corrupt set, all initially set to  $\perp$  and to be used in the main construction. If both  $(y_j, y_k)$  are  $\perp$ , then the corrupt set will be non-empty. The garblers output their corrupt set. We now prove a few lemmas. The protocol  $\text{god}_i$  appears in Figure 8.

**Lemma 15.**  $P_\beta \notin \mathcal{C}_\alpha$  holds for honest  $P_\alpha, P_\beta$ .

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

**Lemma 16.** Assuming input consistency, at the end of protocol  $\text{god}_i$ , an honest evaluator  $P_i$  either computes the output or identifies the corrupt party.

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

- $P_k$  **sends nOK:** If  $P_j$  sends nOK too,  $P_i$  receives  $x_l$  from  $P_l$  for  $l \in \{j, k\}$  (else  $P_k$  is identified to be corrupt) and computes  $f$  on inputs  $x_i, x_j, x_k$ . If  $P_j$  sends OK, then the garbled circuit  $\mathcal{C}_k$  is correctly constructed and the corresponding encoding

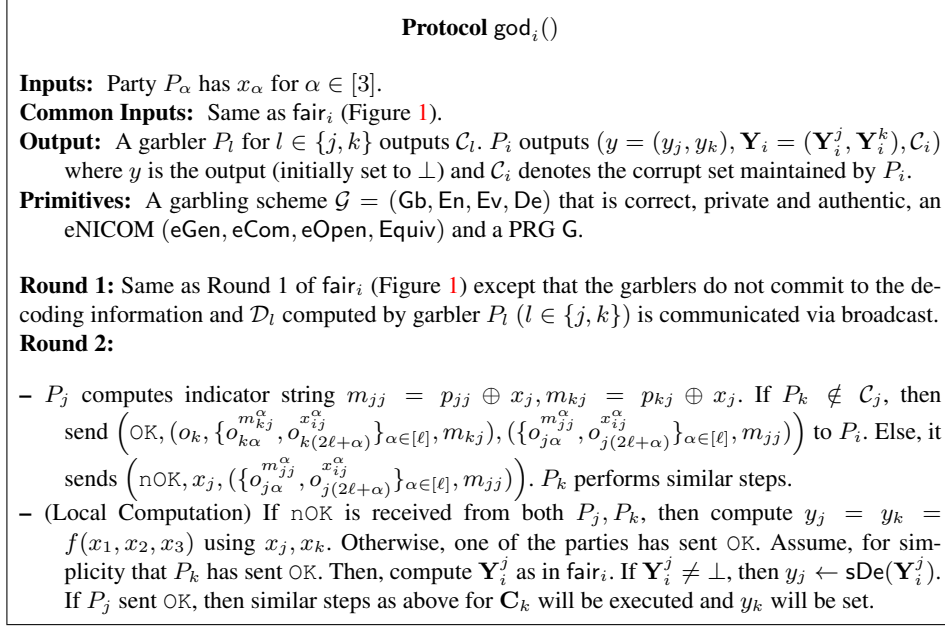


Fig. 8: Protocol  $\text{god}_i$

information is correctly committed. The only way a corrupt garbler  $P_k$  can stop  $P_i$  from evaluating  $\mathcal{C}_k$  (and avoid being caught by  $P_i$ ) is by sending encoded inputs corresponding to  $(x_k, x_{ik})$  that are inconsistent with  $\mathcal{C}_k$  via breaching the binding property of NICOM which happens only with negligible probability.

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

## 5.2 Protocol $\text{god}$

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



The cases when a corrupt  $P_i$  is detected by the end of first round itself, the honest party who makes the identification, halts the execution where it plays the evaluator with the corrupt set as the output and also halts  $\text{god}_i$  to stop letting  $P_i$  get output in  $\text{god}_i$ . Since the detection may be owing to non-commitment of any input by  $P_i$  in  $\text{god}_i$ , the unique input of  $P_i$  has to be set to the one that it commits in the running execution or as a default value when either there is no running execution or  $P_i$  does not commit to anything in the running execution. Specifically, if both the honest parties identify  $P_i$  to be corrupt by the end of first round, both would have exchanged their input as per the code of god protocols and a default common value is taken as the input of  $P_i$  to compute the function output by the end of second round itself and the output is handed over to  $P_i$  in third round. Handing the output to corrupt  $P_i$  is necessary to technically realise the functionality correctly where the corrupt party also gets the output. If just one of the parties detects the corrupt party  $P_i$ , say  $P_j$ , it stops its execution as the evaluator in  $\text{god}_j$  and as garbler in  $\text{god}_i$  to prevent  $P_i$  getting any output in  $\text{god}_i$ . Now  $P_i$  has two options: either it passes on its input on clear to  $P_k$  or it lets  $P_k$  to evaluate the garbled circuit of  $P_j$  by giving its encoded input. In either case, this input of  $P_i$  is taken as his committed input and the output computed by  $P_k$  is the one to be outputted by all. (Note that  $P_i$ 's own GC will not be approved by its co-garbler who has identified it as corrupt by the end of first round.)  $P_k$  can simply pass on the output to  $P_i$  and  $P_j$  in the third round and  $P_j$  simply takes the output of  $P_k$  who it knows to be honest. Our protocol appears in Figure 9. The proof of correctness appear below and the full proof in Appendix F.

**Lemma 17.**  $P_\beta \notin \mathcal{C}_\alpha$  holds for honest  $P_\alpha, P_\beta$  in protocol  $\text{god}_i$ , where  $i \in [3]$ .

*Proof.* This lemma follows from Lemma 15 and the fact that the following will not be true for honest  $(P_\alpha, P_\beta)$ : (a)  $P_\beta$  sends  $o_{\beta\alpha}, x_{\beta\alpha}$  to  $P_\alpha$  such that  $\text{Open}(c_{\beta\alpha}, o_{\beta\alpha}) \neq x_{\beta\alpha}$  (b) Both  $P_\alpha, P_\beta$  are garblers and  $p_{\beta\beta} \neq x_{\beta\alpha}$ . (c)  $P_\beta$  is the garbler,  $P_\alpha$  is an evaluator and  $m_{\beta\beta} \neq x_{\beta\alpha}$   $\square$

**Lemma 18.** Every party  $P_i$  uses its ‘committed’ input  $x_i$  (Definition 1) in its GCs in  $\{\text{god}_j, \text{god}_k\}$ . Otherwise, it is identified by at least one of the honest parties.

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

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

- belongs to the corrupt set of both the honest parties,
- belongs to the corrupt set of exactly one of the honest parties and
- does not belong to the corrupt set of the honest parties

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

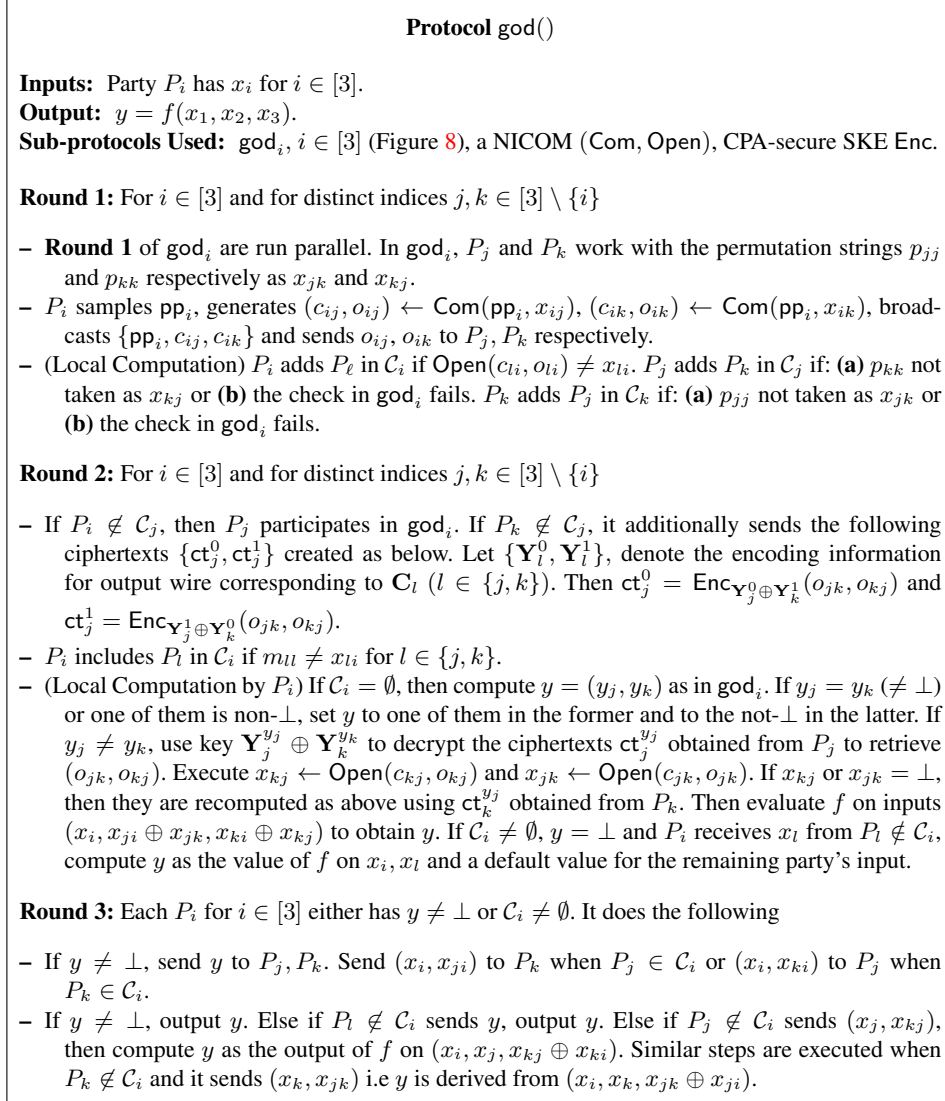


Fig. 9: A Three-Round 3PC protocol achieving guaranteed output delivery

**Lemma 19.** *Assuming that the corrupt party belongs to the corrupt set of both the honest parties by the end of the first round, protocol god is correct.*

*Proof.* In this case,  $P_i$  and  $P_j$  does not communicate at all in the second round of  $\text{god}_k$  preventing  $P_k$  to compute an output. In  $\text{god}_i$  and  $\text{god}_j$ ,  $P_j$  and  $P_i$ , respectively send their inputs on clear to each other along with  $\text{nOK}$  signal. Both compute  $y$  on the inputs  $x_i, x_j$  that are exchanged and a default common value for  $x_k$  by the end of round 2. In the third round,  $P_k$  receives  $y$  from the honest parties and the honest parties output  $y$ .

In this case the unique input of the corrupt party taken for computation is the default commonly-agreed value.  $\square$

**Lemma 20.** *Assuming that the corrupt party belongs to the corrupt set of exactly one of the honest parties by the end of the first round, protocol  $\text{god}$  is correct.*

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

**Lemma 21.** *Assuming that the corrupt party does not belong to the corrupt set of both the honest parties by the end of the first round, protocol  $\text{god}$  is correct.*

*Proof.* In this case,  $P_k$  must have ‘committed’ (Definition 1) to his input (else would be identified by atleast one of the honest parties at end of Round 1) and obtained output  $y$  based on its committed input during  $\text{god}_k$ . Further,  $P_k$  is not detected yet by the end of first round, implies that it has played the role of the garblers in  $\text{god}_i$  and  $\text{god}_j$  honestly in the first round. In this case, we prove that no matter how  $P_k$  behaves in the second round, the honest parties will obtain  $y$  based on their inputs and  $P_k$ ’s committed input. We present the argument for honest  $P_i$ . Similar argument holds for  $P_j$ . Based on the observation that  $P_i$  must have attempted to evaluate  $\mathcal{C}_k$  since  $P_j$  must have sent  $\text{OK}$  signal in  $\text{god}_i$ , we consider the following cases:

- $P_i$  is unsuccessful in evaluating the circuit  $\mathcal{C}_k$  of garbler  $P_k$  in  $\text{god}_i$ . This implies  $P_k$  has given inconsistent encoded input for its circuit to  $P_i$ . So  $P_i$  concludes the second round with  $y = \perp$  and  $\mathcal{C}_i = P_k$ .
- $P_i$  is successful in evaluating the circuit  $\mathcal{C}_k$  of garbler  $P_k$  in  $\text{god}_i$ . By Lemma 18,  $P_k$  must have given encoded input corresponding to its committed input  $x_k$  for  $\mathcal{C}_k$ . This implies the output obtained via  $\mathcal{C}_k$  (i.e  $y_k$ ) is the desired  $y$  in this case. Now we have two cases based on whether  $P_k$  approves the garbled circuit constructed by  $P_j$  or not. In each case we show that,  $P_i$  outputs the desired  $y$  by the end of second round itself. If  $P_k$  disapproves, then  $y_j = \perp$  and  $P_i$  outputs the value  $y = y_k$  obtained via the GC  $\mathcal{C}_k$  as per the specification of  $\text{god}_i$ . Otherwise,  $P_i$  evaluates both

circuits, namely  $C_j$  and  $C_k$ . If the outputs are the same, then the guarantee provided by Lemma 18 implies  $P_i$  outputs the desired  $y$ . Else if  $P_i$  has got conflicting outputs ( $y_j \neq y_k$ ), then it gets access to the key  $Y_j^{y_j} \oplus Y_k^{y_k}$  and uses it to decrypt at least one of the ciphertexts  $\{ct_j^{y_j}, ct_k^{y_k}\}$  generated by  $P_j$  and  $P_k$ . If the decryption of only the honest party  $P_j$ 's ciphertext succeeds, then  $P_i$  obtains  $(o_{jk}, o_{kj})$ , retrieves his missing shares  $x_{jk}, x_{kj}$  and computes  $y$  using  $x_i, x_j = x_{ji} \oplus x_{jk}$  and  $x_k = x_{ki} \oplus x_{kj}$  where  $P_i$  and  $P_j$  receives  $x_{ki}$  and  $x_{kj}$  respectively from  $P_k$  in  $\text{god}_k$ . Even if corrupt  $P_k$ 's ciphertext is decrypted successfully, the  $y$  computed is still as above due the fact that  $P_k$  cannot open a different value for  $x_{jk}, x_{kj}$  due to the binding property of NICOM.  $P_i$  retains this output in the third round.

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

**Theorem 3.** *Protocol god is correct.*

*Proof.* The proof follows from Lemma 19,20,21 as we have considered all the cases exhaustively based on whether the corrupt party  $P_k$  is identified by none, exactly one or both the honest parties by the end of first round.  $\square$

## 6 Lower Bounds

In this paper, we present two lower bounds— **(a)** three rounds are necessary for achieving fairness in the presence of pair-wise private channels and a broadcast channel; **(b)** three rounds are necessary for achieving unanimous abort in the presence of just pair-wise private channels (and no broadcast). The second result holds even if broadcast was allowed in the first round. Our results extend for any  $n$  and  $t$  with  $3t \geq n > 2t$  via standard player-partitioning technique [Lyn96]. Our results imply the following. First, selective abort is the best amongst the four notions (considered in this work) that we can achieve in two rounds without broadcast (from **(b)**). Second, unanimous abort as well as fairness require 3 rounds in the absence of broadcast (from **(b)**). Third, broadcast does not help to improve the round complexity of fairness (from **(a)**). Lastly, guaranteed output delivery requires 3 rounds with broadcast (from **(a)**).

### 6.1 The Impossibility of 2-round Fair 3PC

In this section, we show that it is impossible to construct a fair 2-round 3PC for general functions. [GLS15] presents a lower bound of three rounds assuming *non-private* point-to-point channels and a broadcast channel (their proof crucially relies on the assumption of non-private channels). [GIKR02] presents a three-round lower bound for fair MPC with  $t \geq 2$  (arbitrary number of parties) in the same network setting as ours. Similar to the lower bounds of [GLS15] and [GIKR02] (for the function of conjunction

of two input bits), our lower bound result does not exploit the rushing nature of the adversary and hence holds for non-rushing adversary as well. Finally, we observe that the impossibility of 2-round 3PC for the information-theoretic setting follows from the impossibility of 2-round 3-party statistical VSS of [PCRR09] (since VSS is a special case of MPC). We now prove the impossibility formally.

**Theorem 4.** *There exist functions  $f$  such that no two-round fair 3PC protocol can compute  $f$ , even in the honest majority setting and assuming access to pairwise-private and broadcast channel.*

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

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

At a high level, we discuss two adversarial strategies  $\mathcal{A}_1$  and  $\mathcal{A}_2$  of  $\mathcal{A}$ . We consider party  $P_i$  launching  $\mathcal{A}_i$  in execution  $\Sigma_i$  ( $i \in [2]$ ) of  $\pi$ . Both the executions are assumed to be run for the same input tuple  $(x_1, x_2, x_3)$  and the same random inputs  $(r_1, r_2, r_3)$  of the three parties. (Same random inputs are considered for simplicity and without loss of generality. The same arguments hold for distribution ensembles as well.) When strategy  $\mathcal{A}_1$  is launched in execution  $\Sigma_1$ , we would claim that by correctness of  $\pi$ ,  $\mathcal{A}$  corrupting  $P_1$  should learn the output  $y = f(x_1, x_2, x_3)$ . Here, we note that the value of  $f(x_1, x_2, x_3)$  depends only on the inputs of honest  $P_2, P_3$  (i.e input values  $x_2, x_3$ ) and is thus well-defined. We refer to  $f(x_1, x_2, x_3)$  as the value determined by this particular combination of inputs  $(x_2, x_3)$  henceforth. Now, since  $\mathcal{A}$  corrupting  $P_1$  learnt the output, due to fairness,  $P_2$  should learn the output too in  $\Sigma_1$ . Next strategy  $\mathcal{A}_2$  is designed so that  $P_2$  in  $\Sigma_2$  can obtain the same view as in  $\Sigma_1$  and therefore it gets the output too. Due to fairness, we can claim that  $P_3$  receives the output in  $\Sigma_2$ . A careful observation then lets us claim that  $P_3$  can, in fact, learn the output at the end of Round 1 itself in  $\pi$ . Lastly, using the above observation, we show a strategy for  $P_3$  that explicitly allows  $P_3$  to breach privacy.

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

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

$\mathcal{A}_2$ :  $P_2$  behaves honestly towards  $P_3$  in Round 1, i.e sends the messages  $p_{2 \rightarrow 3}^1, b_2^1$  according to the protocol specification. However  $P_2$  does not communicate to  $P_1$  in Round 1. In Round 2,  $P_2$  waits to receive messages from  $P_3$ , but does not communicate to the other parties.

Next we present the views of the parties in the two executions  $\Sigma_1$  and  $\Sigma_2$  in Table 1. The communications that could potentially be different from the communications

in an honest execution (where all parties behave honestly) with the considered inputs and random inputs of the parties are appended with  $\star$  (e.g.  $p_{1 \rightarrow 3}^2(\star)$ ). We now prove a sequence of lemmas to complete our proof.

Table 1: Views of  $P_1, P_2, P_3$  in  $\Sigma_1$  and  $\Sigma_2$

	$\Sigma_1$			$\Sigma_2$		
	$V_1$	$V_2$	$V_3$	$V_1$	$V_2$	$V_3$
Initial Input	$(x_1, r_1)$	$(x_2, r_2)$	$(x_3, r_3)$	$(x_1, r_1)$	$(x_2, r_2)$	$(x_3, r_3)$
Round 1	$p_{2 \rightarrow 1}^1, p_{3 \rightarrow 1}^1$ $b_2^1, b_3^1$	$p_{1 \rightarrow 2}^1, p_{3 \rightarrow 2}^1$ $b_1^1, b_3^1$	$p_{1 \rightarrow 3}^1, p_{2 \rightarrow 3}^1$ $b_1^1, b_2^1$	$-, p_{3 \rightarrow 1}^1$ $b_2^1, b_3^1$	$p_{1 \rightarrow 2}^1, p_{3 \rightarrow 2}^1$ $b_1^1, b_3^1$	$p_{1 \rightarrow 3}^1, p_{2 \rightarrow 3}^1$ $b_1^1, b_2^1$
Round 2	$p_{2 \rightarrow 1}^2, p_{3 \rightarrow 1}^2$ $b_2^2, b_3^2$	$-, p_{3 \rightarrow 2}^2$ $b_3^2$	$-, p_{2 \rightarrow 3}^2$ $b_2^2$	$-, p_{3 \rightarrow 1}^2$ $b_3^2$	$p_{1 \rightarrow 2}^2(\star), p_{3 \rightarrow 2}^2$ $b_1^2(\star), b_3^2$	$-, p_{1 \rightarrow 3}^2(\star)$ $b_1^2(\star)$

**Lemma 22.** A corrupt  $P_1$  launching  $\mathcal{A}_1$  in  $\Sigma_1$  should learn the output  $y = f(x_1, x_2, x_3)$ .

*Proof.* The proof follows easily. Since  $P_1$  behaved honestly during Round 1, it received all the desired communication from honest  $P_2$  and  $P_3$  in Round 2 (refer to Table 1 for the view of  $P_1$  in  $\Sigma_1$  in the end of Round 2). So it follows from the correctness property that his view at the end of the protocol i.e  $V_1$  should enable  $P_1$  to learn the correct function output  $f(x_1, x_2, x_3)$ .  $\square$

**Lemma 23.** A corrupt  $P_2$  launching  $\mathcal{A}_2$  in  $\Sigma_2$  should learn the output  $y$ .

*Proof.* We prove the lemma with the following two claims. First, the view of  $P_2$  in  $\Sigma_2$  subsumes the view of honest  $P_2$  in  $\Sigma_1$ . Second,  $P_2$  learns the output in  $\Sigma_1$  due to the fact that the corrupt  $P_1$  learns it and  $\pi$  is fair. We now prove our first claim. In  $\Sigma_1$ , we observe that  $P_2$  has received communication from both  $P_1$  and  $P_3$  in the first round, and only from  $P_3$  in the second round. So  $V_2 = \{x_2, r_2, p_{1 \rightarrow 2}^1, b_1^1, p_{3 \rightarrow 2}^1, b_3^1, p_{3 \rightarrow 2}^2, b_3^2\}$  (refer to Table 1). We now analyze  $P_2$ 's view in  $\Sigma_2$ . Both  $P_1$  and  $P_3$  are honest and must have sent  $\{p_{1 \rightarrow 2}^1, b_1^1, p_{3 \rightarrow 2}^1, b_3^1\}$  according to the protocol specifications in Round 1. Since  $P_3$  received the expected messages from  $P_2$  in Round 1,  $P_3$  must have sent  $\{p_{3 \rightarrow 2}^2, b_3^2\}$  in Round 2. Note that we can rule out the possibility of  $P_3$ 's messages in this round having been influenced by  $P_1$  possibly reporting  $P_2$ 's misbehavior towards  $P_1$ . This holds since  $P_3$  would send the messages in the beginning of Round 2. We do not make any assumption regarding  $P_1$ 's communication to  $P_2$  in Round 2 since  $P_1$  has not received the expected message from  $P_2$  in Round 1. Thus, overall,  $P_2$ 's view  $V_2$  comprises of  $\{x_2, r_2, p_{1 \rightarrow 2}^1, b_1^1, p_{3 \rightarrow 2}^1, b_3^1, p_{3 \rightarrow 2}^2, b_3^2\}$  (refer to Table 1). Note that there may also be some additional messages from  $P_1$  to  $P_2$  in Round 2 which can be ignored by  $P_2$ . These are marked with ' $\star$ ' in Table 1. A careful look shows that the view of  $P_2$  in  $\Sigma_2$  subsumes the view of honest  $P_2$  in  $\Sigma_1$ . This concludes our proof.  $\square$

**Lemma 24.**  $P_3$  in  $\Sigma_2$  should learn the output  $y$  by the end of Round 1.

*Proof.* According to the previous lemma,  $P_2$  should learn the function output in  $\Sigma_2$ . Due to fairness property, it must hold that an honest  $P_3$  learns the output as well (same

as obtained by  $P_2$  i.e  $y$  with respect to  $x_2$ ). First, we note that as per strategy  $\mathcal{A}_2$ ,  $P_2$  only communicates to  $P_3$  in Round 1. Second, we argue that the second round communication from  $P_1$  does not impact  $P_3$ 's output computation as follows.

We observe that the function output depends only on  $(x_2, x_3)$ . Clearly, Round 1 messages  $\{p_{1 \rightarrow 3}^1, b_1^1\}$  of  $P_1$  does not depend on  $x_2$ . Next, since there is no private communication to  $P_1$  from  $P_2$  as per strategy  $\mathcal{A}_2$ , the only information that can possibly hold information on  $x_2$  and can impact the round 2 messages of  $P_1$  is  $b_2^1$ . However, since this is a broadcast message,  $P_3$  holds this by the end of Round 1 itself.  $\square$

**Lemma 25.** *A corrupt  $P_3$  violates the privacy property of  $\pi$ .*

*Proof.* The adversary corrupting  $P_3$  participates in the protocol honestly by fixing input  $x_3 = 0$ . Since  $P_3$  can get the output from  $P_2$ 's and  $P_1$ 's round 1 communication (Lemma 24), it must be true that  $P_3$  can evaluate the function  $f$  locally by plugging in any value of  $x_3$ . (Note that  $P_2$  and  $P_1$ 's communication in round 1 are independent of the communication of  $P_3$  in the same round.) Now a corrupt  $P_3$  can plug in  $x_3 = 1$  locally and learn  $x_2$  (via the output  $x_2 \wedge x_3$ ). In the ideal world, corrupt  $P_3$  must learn nothing beyond the output 0 as it has participated in the protocol with input 0. But in the execution of  $\pi$  (in which  $P_3$  participated honestly with input  $x_3 = 0$ ),  $P_3$  has learnt  $x_2$ . This is a clear breach of privacy as  $P_3$  learns  $x_2$  regardless of his input.  $\square$   
Hence, we have arrived at a contradiction, completing the proof of Theorem 4.  $\square$

## 6.2 The Impossibility of 2-round 3PC with Unanimous Abort

**Theorem 5.** *There exist functions  $f$  such that no two-round 3PC protocol achieving security with unanimous abort can compute  $f$  assuming access to pairwise-private channels, even in the honest majority setting.*

*Proof.* We prove the theorem by contradiction. We assume that there exists a two-round 3PC protocol  $\pi$  achieving security with unanimous abort that can compute the same function  $f(x_1, x_2, x_3)$  considered in the proof of Theorem 4.

At a high level, we discuss three adversarial strategies  $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$  of  $\mathcal{A}$ . We consider party  $P_1$  launches  $\mathcal{A}_1$  in execution  $\Sigma_1$ , and  $P_2$  launches  $\mathcal{A}_2, \mathcal{A}_3$  in executions  $\Sigma_2, \Sigma_3$  of  $\pi$  respectively. For the sake of simplicity, the executions are assumed to be run for the same input tuple  $(x_1, x_2, x_3)$  and the same random inputs  $(r_1, r_2, r_3)$  (without loss of generality) of the three parties. We use the notation  $V_i^j$  to denote the view of party  $P_i$  at the end of execution  $\Sigma_j$  of  $\pi$ . The skeleton of the proof goes as follows: We first claim that strategy  $\mathcal{A}_1$  leads to honest  $P_2$  computing the output  $y = f(x_1, x_2, x_3)$ . Here, we note that the value of  $f(x_1, x_2, x_3)$  depends only on the inputs of honest  $P_2, P_3$  (i.e input values  $x_2, x_3$ ) and is thus well-defined. We refer to  $f(x_1, x_2, x_3)$  as the value determined by this particular combination of inputs  $(x_2, x_3)$  henceforth. Since the protocol achieves unanimous abort, honest  $P_3$ 's view  $V_3^1$  at the end of  $\Sigma_1$  must lead to output computation of  $y$  by  $P_3$ . Next, strategy  $\mathcal{A}_2$  executed by  $P_2$  during  $\Sigma_2$  results in  $P_3$  having the same view as in  $\Sigma_1$  i.e  $V_3^1 = V_3^2$ . Thus, honest  $P_3$  computes the output and to preserve the property of unanimous abort, honest  $P_1$  with view  $V_1^2$  must also compute the output. Finally, we present a strategy  $\mathcal{A}_3$  by  $P_2$  during  $\Sigma_3$  that results in

$P_1$  having the same view as in  $\Sigma_2$  i.e  $V_1^2 = V_1^3$ . It follows that honest  $P_1$  computes the output and therefore honest  $P_3$  with view  $V_3^3$  must be able to compute the output too. This results in a contradiction as we conclude that if  $P_3$ 's view  $V_3^3$  enables output computation,  $P_3$  must be able to compute the output at the end of Round 1 itself which violates privacy as proved in Lemma 25.

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

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

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

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

Next we present the views of the parties in  $\Sigma_1, \Sigma_2$  and  $\Sigma_3$  in Table 2. Here,  $\overline{p_{1 \rightarrow 3}^2}$  is the message that should be sent by  $P_1$  to  $P_3$  according to the protocol incase  $P_1$  didn't receive anything from  $P_2$  in Round 1. Besides this, the communications that could potentially be different from the communications in an honest execution with the considered inputs and random inputs of the parties are appended with  $\star$  (e.g.  $p_{1 \rightarrow 2}^2(\star)$ ). We now prove a sequence of lemmas to complete our proof.

Table 2: Views of  $P_1, P_2, P_3$  in  $\Sigma_1, \Sigma_2, \Sigma_3$

	$\Sigma_1$			$\Sigma_2$			$\Sigma_3$		
	$V_1$	$V_2$	$V_3$	$V_1$	$V_2$	$V_3$	$V_1$	$V_2$	$V_3$
Initial Input	$(x_1, r_1)$	$(x_2, r_2)$	$(x_3, r_3)$	$(x_1, r_1)$	$(x_2, r_2)$	$(x_3, r_3)$	$(x_1, r_1)$	$(x_2, r_2)$	$(x_3, r_3)$
Round 1	$p_{2 \rightarrow 1}^1, p_{3 \rightarrow 1}^1$	$p_{1 \rightarrow 2}^1, p_{3 \rightarrow 2}^1$	$p_{1 \rightarrow 3}^1, p_{2 \rightarrow 3}^1$	$\neg, p_{3 \rightarrow 1}^1$	$p_{1 \rightarrow 2}^1, p_{3 \rightarrow 2}^1$	$p_{1 \rightarrow 3}^1, p_{2 \rightarrow 3}^1$	$\neg, p_{3 \rightarrow 1}^1$	$p_{1 \rightarrow 2}^1, p_{3 \rightarrow 2}^1$	$p_{1 \rightarrow 3}^1, p_{2 \rightarrow 3}^1$
Round 2	$p_{2 \rightarrow 1}^2, p_{3 \rightarrow 1}^2$	$p_{1 \rightarrow 2}^2, p_{3 \rightarrow 2}^2$	$\overline{p_{1 \rightarrow 3}^2}, p_{2 \rightarrow 3}^2$	$\neg, p_{3 \rightarrow 1}^2$	$p_{1 \rightarrow 2}^2(\star), p_{3 \rightarrow 2}^2$	$\overline{p_{1 \rightarrow 3}^2}, p_{2 \rightarrow 3}^2$	$\neg, p_{3 \rightarrow 1}^2$	$p_{1 \rightarrow 2}^2(\star), p_{3 \rightarrow 2}^2$	$\overline{p_{1 \rightarrow 3}^2}, \neg$

**Lemma 26.**  $P_3$  computes the output  $y = f(x_1, x_2, x_3)$  at the end of  $\Sigma_1$ .

*Proof.* The proof follows easily. During  $\Sigma_1$ , as per strategy  $\mathcal{A}_1$ , corrupt  $P_1$  behaved honestly to  $P_2$  throughout  $\pi$ . Therefore  $P_2$  would compute the output  $y = f(x_1, x_2, x_3)$ . Due to property of unanimous abort, honest  $P_3$  must learn the output as well.  $\square$

**Lemma 27.**  $P_3$  computes the output  $y = f(x_1, x_2, x_3)$  at the end of  $\Sigma_2$ .

*Proof.* We observe that the view of  $P_3$  during  $\Sigma_1, \Sigma_2$  is same. As per both strategies  $\mathcal{A}_1$ , and  $\mathcal{A}_2$ ,  $P_3$  receives communication from  $P_1, P_2$  as per honest execution in Round 1. In Round 2, according to  $\mathcal{A}_1$ , corrupt  $P_1$  sends  $\overline{p_{1 \rightarrow 3}^2}$  as per protocol specification for case when  $P_1$  receives nothing from  $P_2$  in Round 1. A similar message would be



sent by honest  $P_1$  to  $P_3$  who did not receive anything from  $P_2$  in Round 1 (as per  $\mathcal{A}_2$ ) during  $\Sigma_2$ . It is now easy to check (refer Table 2) that  $V_3^1 = V_3^2$ . Finally, since  $V_3^1$  leads to output computation of  $y$  as per Lemma 26,  $P_3$ 's view at the end of  $\Sigma_2$  i.e  $V_3^2$  must result in  $P_3$  computing the output  $y$ .  $\square$

**Lemma 28.**  $P_3$  learns the output at the end of  $\Sigma_3$ .

*Proof.* Firstly, it follows from lemma 27 and property of unanimous abort that honest  $P_1$  must compute the output at the end of  $\Sigma_2$ . Next, it is easy to check that  $V_1^2 = V_1^3$  (refer Table 2). We can thus conclude that honest  $P_1$  computes the output at the end of  $\Sigma_3$ . Therefore, honest  $P_3$  must also be able to compute the output at the end of  $\Sigma_3$  (by assumption that  $\pi$  achieves unanimous abort).  $\square$

Finally, we now prove that  $P_3$  learns the output at the end of Round 1 (similar to Lemma 24).

**Lemma 29.**  $P_3$  in  $\Sigma_3$  should learn the output  $y$  by the end of Round 1.

*Proof.* According to lemma 28,  $P_3$  should learn the function output in  $\Sigma_3$ . First, we note that as per strategy  $\mathcal{A}_3$ , corrupt  $P_2$  only communicates to  $P_3$  in Round 1. Second, we argue that the second round communication from  $P_1$  does not impact  $P_3$ 's output computation as follows.

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

If  $P_3$  is able to compute output at the end of Round 1, we know that protocol  $\pi$  violates privacy (proved in Lemma 25). We have thus arrived at a contradiction, concluding the proof of Theorem 5.  $\square$

We point that even if broadcast was allowed in the first round, all the above arguments would still hold.

*Alternative functions.* While it suffices to show impossibility with respect to a particular function to rule out the possibility of having generic protocols, we cite yet another function that can lead to the same conclusion. Consider a function  $f'$  that outputs the message  $m$  which is the decryption of ciphertext  $c$  ( $P_2$ 's input) where the decryption key  $k$  constitutes  $P_3$ 's input. All our arguments still hold except Lemma 25: Instead of the argument of how privacy could be breached by corrupt  $P_3$  who gets access to output at the end of Round 1, in the context of this function  $f'$ , a corrupt  $P_3$  (who gets access to the output at the end of Round 1 itself) would be able to get decryptions of the ciphertext  $c$  corresponding to multiple keys  $k$  of his choice which violates correctness.

## 7 Conclusion

In this paper, we settle exact round complexity of 3PC with selective abort, unanimous abort, fairness and guaranteed output delivery in a setting with private pairwise channels and with or without broadcast channel. Our lower bounds extend for any  $n$  and  $t$  with  $3t > n > 2t$ . Our protocols rely on injective OWF.

## References

- AARV17. Benny Applebaum, Barak Arkis, Pavel Raykov, and Prashant Nalini Vasudevan. Conditional disclosure of secrets: Amplification, closure, amortization, lower-bounds, and separations. In *CRYPTO*, 2017.
- ACJ17. Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO*, 2017.
- AFL<sup>+</sup>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 *ACM CCS*, 2016.
- AJL<sup>+</sup>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*, 2012.
- AMPR14. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT*, 2014.
- BCD<sup>+</sup>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*, 2009.
- BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.
- Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
- BFO12. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *CRYPTO*, 2012.
- BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.
- BH06. Zuzana Beerliová-Trubníová and Martin Hirt. Efficient multi-party computation with dispute control. In *TCC*, 2006.
- BHP17. Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *TCC*, 2017.
- BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, 2012.
- BKP11. Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *ASIACRYPT*, 2011.
- BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security- ESORICS*, 2008.
- BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *ACM STOC*, 1990.
- BTW12. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In *FC*, 2012.
- Can00. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1), 2000.
- CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *ACM STOC*, 1988.
- CDD<sup>+</sup>99. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, 1999.

- CDG87. David Chaum, Ivan Damgård, and Jeroen Graaf. Multiparty computations ensuring privacy of each party’s input and correctness of the result. In *CRYPTO*, 1987.
- CGMV17. Nishanth Chandran, Juan A. Garay, Payman Mohassel, and Satyanarayana Vusirikala. Efficient, constant-round and actively secure MPC: beyond the three-party case. In *ACM CCS*, 2017.
- CHOR16. Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. Characterization of secure multiparty computation without broadcast. In *TCC*, 2016.
- CIO98. Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *ACM STOC*, 1998.
- CKMZ14. Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *CRYPTO*, 2014.
- CL14. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT*, 2014.
- Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *ACM STOC*, 1986.
- CMF<sup>+</sup>14. Koji Chida, Gembu Morohashi, Hitoshi Fuji, Fumihiko Magata, Akiko Fujimura, Koki Hamada, Dai Ikarashi, and Ryuichi Yamamoto. Implementation and evaluation of an efficient secure computation system using ‘R’ for healthcare statistics. *Journal of the American Medical Informatics Association*, 2014.
- DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, 2007.
- DO10. Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, 2010.
- DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
- 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*, 2017.
- FNO15. Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT*, 2015.
- Gei07. Martin Geisler. Viff: Virtual ideal functionality framework, 2007.
- GGHR14. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, 2014.
- GGM86. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *JACM*, 1986.
- GIKM00. Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 2000.
- GIKR01. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *ACM STOC*, 2001.
- GIKR02. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *CRYPTO*, 2002.
- GL02. Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *DISC*, 2002.
- GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, 2015.
- GMPP16. Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT*, 2016.
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM STOC*, 1987.

- Gol01. Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.
- HHPV17. Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkatasubramanian. Round-optimal secure multi-party computation. *Cryptology ePrint Archive*, Report 2017/1056, 2017. <https://eprint.iacr.org/2017/1056>.
- HKK<sup>+</sup>14. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In *CRYPTO*, 2014.
- HLP11. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, 2011.
- IKKP15. Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO*, 2015.
- IKP10. Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, 2010.
- IKP<sup>+</sup>16. Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *CRYPTO*, 2016.
- IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, 2008.
- IW14. Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *ICALP 2014*, 2014.
- JKO13. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *CCS*, 2013.
- JW16. Zahra Jafargholi and Daniel Wichs. Adaptive security of yao’s garbled circuits. In *TCC 2016-B*, 2016.
- KO04. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO*, 2004.
- KS05. Jonathan Katz and Ji Sun Shin. Modeling insider attacks on group key-exchange protocols. In *CCS*, 2005.
- KS06. Mehmet S Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of yao’s garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, 2006.
- KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, 2008.
- LADM14. John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-scale secure multiparty computation. In *ESOP*, 2014.
- LDDA12. John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In *ACM SIGPLAN ICFP’12*, 2012.
- Lin13. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.
- Lin17. Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. 2017.
- LP07. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.
- LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yao’s protocol for two-party computation. *J. Cryptology*, 2009.
- Lyn96. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
- MF06. Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, 2006.
- MR17. Payman Mohassel and Mike Rosulek. Non-interactive secure 2pc in the offline/online and batch settings. In *EUROCRYPT*, 2017.

- MRZ15. Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *ACM CCS*, 2015.
- MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, 2016.
- Nao91. Moni Naor. Bit commitment using pseudorandomness. *J. Cryptology*, 4(2), 1991.
- PCRR09. Arpita Patra, Ashish Choudhary, Tal Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In *CRYPTO*, 2009.
- Ped91. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.
- RB89. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *ACM STOC*, 1989.
- RR16. Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *USENIX Security Symposium*, 2016.
- SS13. Abhi Shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In *ACM CCS*, 2013.
- Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982.
- ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015.

# Supplementary Material

## A Primitives

### A.1 Properties of Garbling Scheme

**Definition 2.** (*Correctness*) A garbling scheme  $\mathcal{G}$  is *correct* if for all input lengths  $n \leq \text{poly}(\kappa)$ , circuits  $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$  and inputs  $x \in \{0, 1\}^n$ , the following probability is negligible in  $\kappa$ :

$$\Pr(\text{De}(\text{Ev}(\mathbf{C}, \text{En}(e, x)), d) \neq C(x) : (\mathbf{C}, e, d) \leftarrow \text{Gb}(1^\kappa, C)).$$

**Definition 3.** (*Privacy*) A garbling scheme  $\mathcal{G}$  is *private* if for all input lengths  $n \leq \text{poly}(\kappa)$ , circuits  $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$ , there exists a PPT simulator  $\mathcal{S}_{\text{priv}}$  such that for all inputs  $x \in \{0, 1\}^n$ , for all probabilistic polynomial-time adversaries  $\mathcal{A}$ , the following two distributions are computationally indistinguishable:

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

**Definition 4.** (*Authenticity*) A garbling scheme  $\mathcal{G}$  is *authentic* if for all input lengths  $n \leq \text{poly}(\kappa)$ , circuits  $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$ , inputs  $x \in \{0, 1\}^n$ , and all PPT adversaries  $\mathcal{A}$ , the following probability is negligible in  $\kappa$ :

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

**Definition 5.** (*Obliviousness*) A garbling scheme  $\mathcal{G}$  achieves *obliviousness* if for all input lengths  $n \leq \text{poly}(\kappa)$ , circuits  $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$ , there exists a PPT simulator  $\mathcal{S}_{\text{obv}}$  such that for all inputs  $x \in \{0, 1\}^n$ , for all probabilistic polynomial-time adversaries  $\mathcal{A}$ , the following two distributions are computationally indistinguishable:

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

### A.2 Non-Interactive Commitment Schemes (NICOM)

*Properties.*

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

$$\left| \Pr_{(c,o) \leftarrow \text{Com}(x)}[\mathcal{A}(c) = 1] - \Pr_{(c,o) \leftarrow \text{Com}(x')}[\mathcal{A}(c) = 1] \right|$$

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

- $\text{Com}(x; r)$ : set  $c = (f(r), x \oplus h(r))$ ; where  $r \in_R \{0, 1\}^n$ ; set  $o = (r, x)$ .
- $\text{Open}(c, o = (r, x))$ : return  $x$  if  $c = (f(r), x \oplus h(r))$ ; otherwise return  $\perp$ .

For commitment of multi-bit string, the Goldreich-Goldwasser-Micali [GGM86] construction from a one-way permutation  $f$  can be used. Recall the GGM construction: given one-way permutation  $f : \{0, 1\}^k \rightarrow \{0, 1\}^k$  with hard-core predicate  $h : \{0, 1\}^k \rightarrow \{0, 1\}$ , first construct a length-doubling pseudorandom generator  $G : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$  via:  $G(s) = f^k(s) \parallel h(f^{k-1}(s)) \dots h(s)$ . Let  $G_0(s)$  denote the first  $k$  bits of  $G(s)$ , and let  $G_1(s)$  denote the last  $k$  bits of  $G(s)$ . For a binary string  $s$ , the commitment  $c$  can be defined as  $c = F(s, 0^\ell) = G_0(\dots(G_0(G_0(s)))\dots)$  with  $o = (s)$ . It is shown in [GGM86] that the function family  $\mathcal{F} = \{F^\kappa\}$  with  $F^\kappa = \{F(s)\}_{s \in \{0, 1\}^\kappa}$  is pseudorandom. Now, note that  $F(s, 0^\ell) = f^{\ell \cdot \kappa}(s)$ . Since  $f$  is a permutation, this means that the function  $g(x) = F(x, 0^\ell)$  is a permutation, and hence the commitment scheme has the binding property. Hiding follows from the property of PRF  $F$  [KS05].

### A.3 Equivocal Non-interactive Commitment Schemes (eNICOM)

An eNICOM comprises of the following algorithms, apart from the ones needed in NICOM:

- $\text{eGen}(1^\kappa)$  returns a public parameter and a corresponding trapdoor  $(\text{epp}, t)$ , where  $\text{epp}$  is used by both  $\text{eCom}$  and  $\text{eOpen}$ . The trapdoor  $t$  is used for equivocation.
- $\text{Equiv}(c, o', x, t)$  is invoked on a certain commitment  $c$  and its corresponding opening  $o'$ , given message  $x$  and the trapdoor  $t$  and returns  $o$  such that  $x \leftarrow \text{eOpen}(\text{epp}, c, o)$ .

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

*Properties.*

- *Correctness:* For all  $(\text{epp}, t) \leftarrow \text{eGen}(1^\kappa)$ ,  $x \in \mathcal{M}$  and  $r \in \mathcal{R}$ , if  $(c, o) \leftarrow \text{eCom}(x; r)$  then  $\text{eOpen}(c, o) = x$ .
- *Binding:* For all  $(\text{epp}, t) \leftarrow \text{eGen}(1^\kappa)$  and for all PPT adversaries  $\mathcal{A}$ , it is with negligible probability that  $\mathcal{A}(\text{epp})$  outputs  $(c, o, o')$  such that  $\text{eOpen}(c, o) \neq \text{eOpen}(c, o')$  and  $\perp \notin \{\text{eOpen}(c, o), \text{eOpen}(c, o')\}$
- *Hiding:* For all  $(\text{epp}, t) \leftarrow \text{eGen}(1^\kappa)$  and for all PPT adversaries  $\mathcal{A}$ , and all  $x, x' \in \mathcal{M}$ , the following difference is negligible:

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

*Instantiations.* We first present the equivocal bit commitment scheme of [CIO98], which is based on Naor’s commitment scheme [Nao91] for single bit message. This scheme avoids the use of public-key primitives. Let  $G : \{0, 1\}^n \rightarrow \{0, 1\}^{4n}$  be a pseudorandom generator.

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

Next, we present the instantiation based on Pedersen commitment scheme [Ped91]. Let  $p, q$  denote large primes such that  $q$  divides  $(p - 1)$ ,  $G_q$  is the unique subgroup of  $\mathbb{Z}_p^*$  of order  $q$  and  $g$  is a generator of  $G_q$ .

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

While in Naor-based instantiation, a specific commitment  $c = G(r_0)$  can be decommitted to either 0 or 1, the Pedersen commitment scheme allows equivocation of *any* commitment. For the purpose of our protocols, even the former weaker property suffices and hence our protocols can be based on symmetric key operations alone.

#### A.4 Symmetric-Key Encryption with Special Correctness

**Definition 6.** A CPA-secure symmetric-key encryption scheme  $\pi = (\text{Gen}, \text{Enc}, \text{Dec})$  satisfies special correctness if there is some negligible function  $\epsilon$  such that for any message  $m$  we have:  $\Pr[\text{Dec}_{k_2}(\text{Enc}_{k_1}(m)) \neq m : k_1, k_2 \leftarrow \text{Gen}(1^\kappa)] \leq \epsilon(\kappa)$

*Instantiation.* Here we present an instantiation borrowed from [JW16, LP09]. Let  $\mathcal{F} = \{f_k\}$  be a family of pseudorandom functions where  $f_k = \{0, 1\}^\kappa \rightarrow \{0, 1\}^{\kappa+s}$ , for  $k \in \{0, 1\}^\kappa$  and  $s$  is a parameter denoting message length.

- $\text{Enc}_k(m) = (r, f_k(r) \oplus m0^\kappa)$  where  $m \in \{0, 1\}^s, r \leftarrow \{0, 1\}^\kappa$  and  $m0^\kappa$  denotes the concatenation of  $m$  with a string of 0s of length  $\kappa$ .
- $\text{Dec}_k(c)$  which parses  $c = (r, z)$ , computes  $w = z \oplus f_k(r)$  and if the last  $\kappa$  bits of  $w$  are 0’s, it outputs the first  $s$  bits of  $w$ , else it outputs  $\perp$

## B The Security Model

We prove the security of our protocols based on the standard real/ideal world paradigm. Essentially, the security of a protocol is analyzed by comparing what an adversary can do in the real execution of the protocol to what it can do in an ideal execution, that is considered secure by definition (in the presence of an incorruptible trusted party). In an



ideal execution, each party sends its input to the trusted party over a perfectly secure channel, the trusted party computes the function based on these inputs and sends to each party its respective output. Informally, a protocol is secure if whatever an adversary can do in the real protocol (where no trusted party exists) can be done in the above described ideal computation. We refer to [Can00, Gol01, Lin17, CL14] for further details regarding the security model.

The “ideal” world execution involves parties  $P_1, P_2, P_3$ , an ideal adversary  $\mathcal{S}$  who may corrupt one of the parties, and a functionality  $\mathcal{F}$ . The “real” world execution involves the PPT parties  $P_1, P_2, P_3$ , and a real world adversary  $\mathcal{A}$  who may corrupt one of the parties. We let  $\text{IDEAL}_{\mathcal{F}, \mathcal{S}}(1^\kappa, z)$  denote the output pair of the honest parties and the ideal-world adversary  $\mathcal{S}$  from the ideal execution with respect to the security parameter  $1^\kappa$  and auxiliary input  $z$ . Similarly, let  $\text{REAL}_{\Pi, \mathcal{A}}(1^\kappa, z)$  denote the output pair of the honest parties and the adversary  $\mathcal{A}$  from the real execution with respect to the security parameter  $1^\kappa$  and auxiliary input  $z$ .

**Definition 7.** For  $n \in \mathbb{N}$ , let  $\mathcal{F}$  be a functionality and let  $\Pi$  be a 3-party protocol. We say that  $\Pi$  securely realizes  $\mathcal{F}$  if for every PPT real world adversary  $\mathcal{A}$ , there exists a PPT ideal world adversary  $\mathcal{S}$ , corrupting the same parties, such that the following two distributions are computationally indistinguishable:

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

**Target Functionalities.** Taking motivation from [CL14, GLS15], we define ideal functionalities  $\mathcal{F}_{\text{sa}}, \mathcal{F}_{\text{ua}}, \mathcal{F}_{\text{fair}}, \mathcal{F}_{\text{god}}$  in Figures 10, 11, 12, 13 for secure 3PC of a function  $f$  with selective abort, unanimous abort, fairness and guaranteed output delivery respectively.

$\mathcal{F}_{\text{sa}}$

**Input:** On message  $(\text{sid}, \text{Input}, x_i)$  from a party  $P_i$  ( $i \in [3]$ ), do the following: if  $(\text{sid}, \text{Input}, *)$  message was received from  $P_i$ , then ignore. Otherwise record  $x'_i = x_i$  internally. If  $x'_i$  is outside of the domain for  $P_i$  ( $i \in [3]$ ), consider  $x'_i = \text{abort}$ .

**Output to adversary:** If there exists  $i \in [3]$  such that  $x'_i = \text{abort}$ , send  $(\text{sid}, \text{Output}, \perp)$  to all the parties. Else, send  $(\text{sid}, \text{Output}, y)$  to the adversary, where  $y = f(x'_1, x'_2, x'_3)$ .

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

Fig. 10: Ideal Functionality for selective abort

$\mathcal{F}_{\text{ua}}$

**Input:** On message  $(\text{sid}, \text{Input}, x_i)$  from a party  $P_i$  ( $i \in [3]$ ), do the following: if  $(\text{sid}, \text{Input}, *)$  message was received from  $P_i$ , then ignore. Otherwise record  $x'_i = x_i$  internally. If  $x'_i$  is outside of the domain for  $P_i$  ( $i \in [3]$ ), consider  $x'_i = \text{abort}$ .

**Output to adversary:** If there exists  $i \in [3]$  such that  $x'_i = \text{abort}$ , send  $(\text{sid}, \text{Output}, \perp)$  to all the parties. Else, send  $(\text{sid}, \text{Output}, y)$  to the adversary, where  $y = f(x'_1, x'_2, x'_3)$ .

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

Fig. 11: Ideal Functionality for unanimous abort

$\mathcal{F}_{\text{fair}}$

**Input:** On message  $(\text{sid}, \text{Input}, x_i)$  from a party  $P_i$  ( $i \in [3]$ ), do the following: if  $(\text{sid}, \text{Input}, *)$  message was received from  $P_i$ , then ignore. Otherwise record  $x'_i = x_i$  internally. If  $x'_i$  is outside of the domain for  $P_i$  ( $i \in [3]$ ), consider  $x'_i = \text{abort}$ .

**Output:** If there exists  $i \in [3]$  such that  $x'_i = \text{abort}$ , send  $(\text{sid}, \text{Output}, \perp)$  to all the parties. Else, send  $(\text{sid}, \text{Output}, y)$  to party  $P_i$  for every  $i \in [3]$ , where  $y = f(x'_1, x'_2, x'_3)$ .

Fig. 12: Ideal Functionality for fairness

$\mathcal{F}_{\text{god}}$

**Input:** On message  $(\text{sid}, \text{Input}, x_i)$  from a party  $P_i$  ( $i \in [3]$ ), do the following: if  $(\text{sid}, \text{Input}, *)$  message was received from  $P_i$ , then ignore. Otherwise record  $x'_i = x_i$  internally. If  $x'_i$  is outside of the domain for  $P_i$ , set  $x'_i$  to be some predetermined default value.

**Output:** Compute  $y = f(x'_1, x'_2, x'_3)$  and send  $(\text{sid}, \text{Output}, y)$  to party  $P_i$  for every  $i \in [3]$ .

Fig. 13: Ideal Functionality for guaranteed output delivery

## C Optimizations

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

protocol independent of the circuit size. Lastly, an optimization with respect to protocol fair is that the inputs to the subprotocol  $\text{cert}_i$  can be modified to hash of the relevant inputs instead, reducing considerably the size of the equality-checking circuit in  $\text{cert}_i$ .

## D Round Optimal 3PC with fairness

### D.1 Schematic Diagram

We present the schematic diagram of the 3-round fair protocol in Figures 14 - 15.

### D.2 Formal Proof of Security for Protocol fair

In this section, we present the proof of security of fair relative to the ideal functionality for fairness shown in Appendix B. For better clarity, we assume without loss of generality that  $P_1$  is corrupt (denoted as  $P_1^*$ ) and describe the simulator  $\mathcal{S}_{\text{fair}}$ . Since the roles of the parties are symmetric in fair, similar proof would hold in case of corrupt  $P_2, P_3$  as well. The simulator plays the role of the honest parties  $P_2, P_3$  and simulates each step of the protocol fair. Recall that during the first two rounds of fair, the two round protocols  $\text{fair}_i$  ( $i \in [3]$ ) and  $\text{cert}_i$  ( $i \in [3]$ ) run in parallel. We divide the description of  $\mathcal{S}_{\text{fair}}$  as follows: We describe  $\mathcal{S}_{\text{fair}}$  during  $\text{fair}_1, \text{cert}_1$  where corrupt  $P_1^*$  is the evaluator and during  $\text{fair}_2, \text{cert}_2$  when corrupt  $P_1^*$  acts as a garbler. The steps corresponding to  $\text{fair}_3$ , and  $\text{cert}_3$  would follow symmetrically from that described corresponding to  $\text{fair}_2, \text{cert}_2$ . Finally, we describe the steps corresponding to the third round. The simulator  $\mathcal{S}_{\text{fair}}$  appears in Figure 16-17 with **R1/R2/R3** indicating simulation for round 1, 2 and 3 respectively and **f/c/F** denoting the steps corresponding to subprotocol  $\text{fair}_i, \text{cert}_i, \text{fair}$  respectively.

When simulating  $\text{fair}_1$ , the simulator does not have access to the inputs of the honest parties. Further, it does not know if and what  $P_1$  commits as its input in Round 1, when simulating and sending the commitments for GC and encoding information in parallel in Round 1. Nor does it know if all the parties will get the output (relative to corrupt  $P_1$ 's committed input from Round 1) or not, when it opens the encoded input and GC in Round 2. The decision comes from  $P_1$ 's behaviour in Round 2. A privacy simulator  $\mathcal{S}_{\text{prv}}$  cannot be invoked for emulating Round 2 message, as  $\mathcal{F}_{\text{fair}}$  cannot be invoked yet and so  $y$  is not available. Instead oblivious simulator  $\mathcal{S}_{\text{obv}}$  is invoked that works without  $y$ . Later if and when  $\mathcal{F}_{\text{fair}}$  is invoked and  $y$  is known,  $\mathcal{S}_{\text{prv}}$  is invoked which simply returns the decoding information that makes the fake GC returned by  $\mathcal{S}_{\text{obv}}$  output  $y$ .

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

- HYB<sub>0</sub>: Same as  $\text{REAL}_{\text{fair}, \mathcal{A}}$ .
- HYB<sub>1</sub>: Same as HYB<sub>0</sub>, except that  $P_2, P_3$  in  $\text{fair}_1$  use uniform randomness rather than pseudo-randomness for the garbled circuit construction.
- HYB<sub>2</sub>: Same as HYB<sub>1</sub>, except that some of the commitments of encoded inputs which will not be sent to  $P_1$  during  $\text{fair}_1$  are replaced with commitments on dummy values. Specifically, these are corresponding to indices not equal to  $m_{22}, m_{23}, x_{12}, x_{13}$  for  $\mathbf{C}_2$  and not equal to  $m_{32}, m_{33}, x_{12}, x_{13}$  for  $\mathbf{C}_3$ .

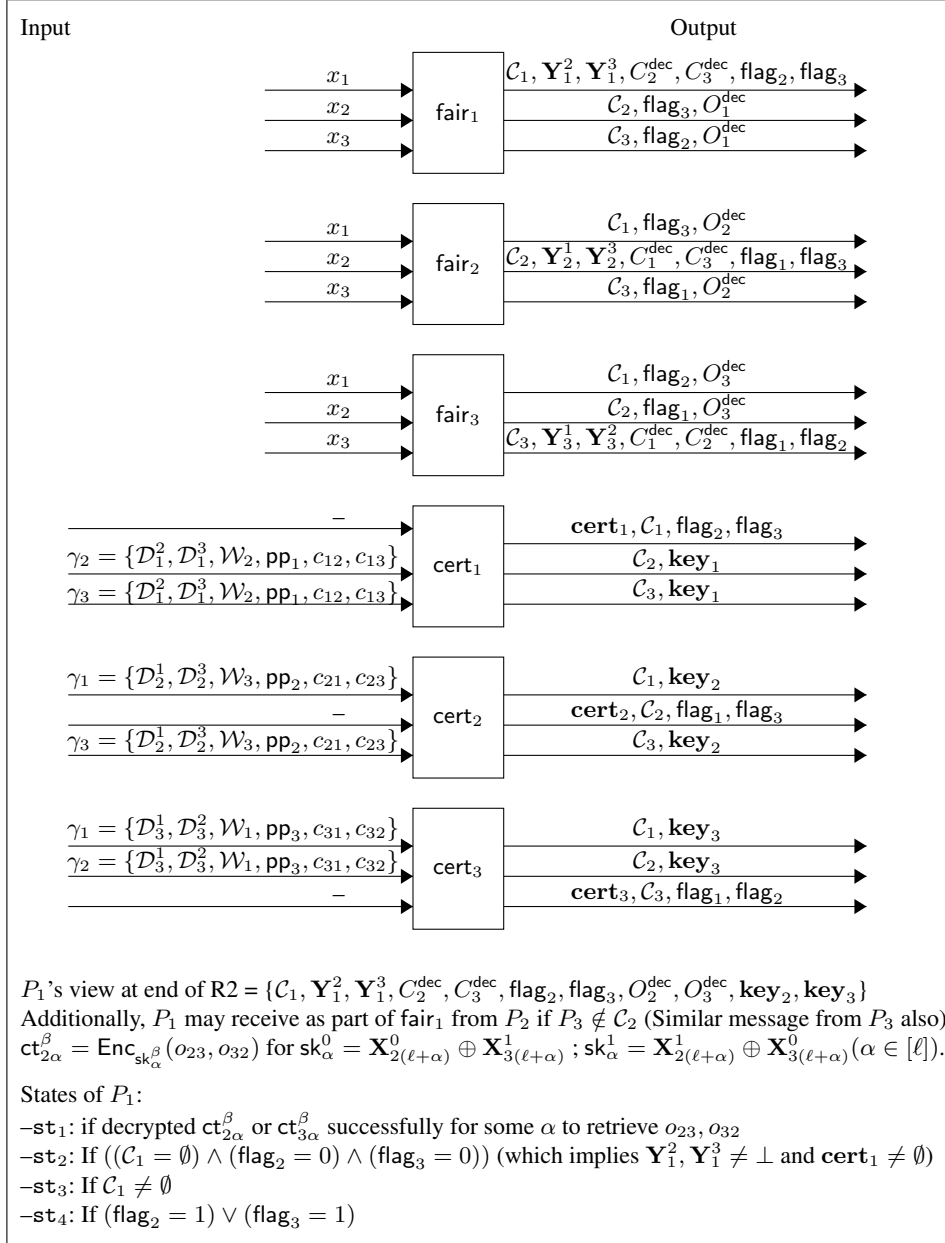
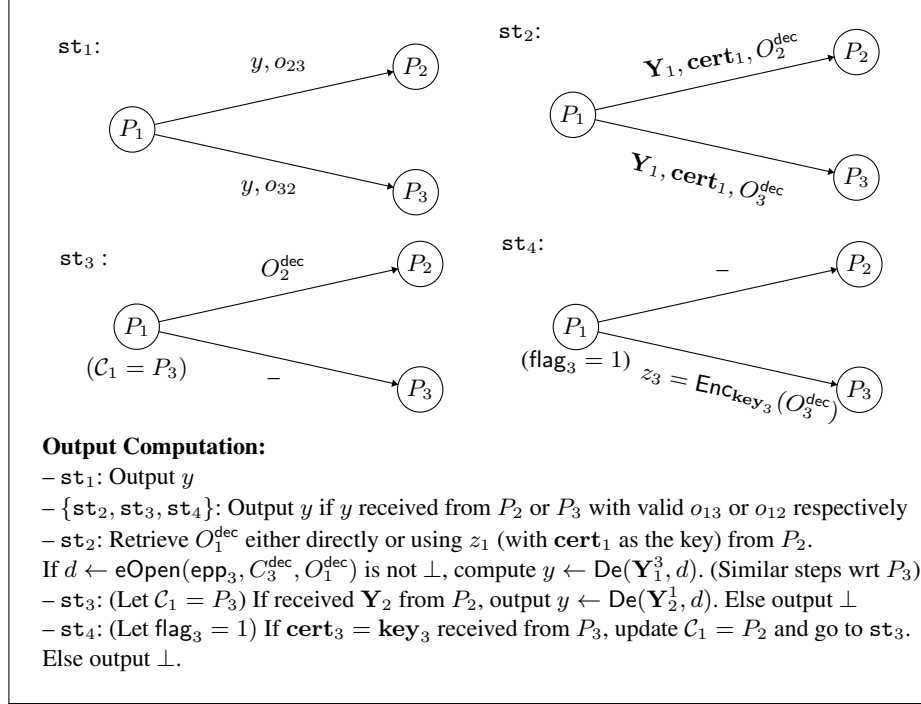


Fig. 14: Schematic Diagram of fair protocol (Round 1 and 2)

- HYB<sub>3</sub> : Same as HYB<sub>2</sub>, except the following:
  - HYB<sub>3.1</sub>: When the execution results in  $P_1$  evaluating GCs during fair<sub>1</sub> but results in abort,  $C_2$  is created as  $C_2' \leftarrow \mathcal{S}_{\text{obv}}(1^\kappa, C, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}}, e_{2(\ell+\alpha)}^{m_{23}}, e_{2(2\ell+\alpha)}^{x_{12}}, e_{2(3\ell+\alpha)}^{x_{13}}\}_{\alpha \in [\ell]})$ . The commitment  $c_2$  is later equiv-

Fig. 15: Schematic Diagram of Protocol fair (Round 3 wrt  $P_1$ )



ocated to  $C'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, C'_2, t_2)$ . The commitment to the decoding information is created for a dummy value. Since the encoding information are committed in round 1 using committing commitments that cannot be equivocated, we invoke  $\mathcal{S}_{\text{obv}}$  using an  $\mathbf{X}$  that corresponds to the correct shares of  $P_1$  and it returns a fake GC (consistent with the labels in  $\mathbf{X}$ ) such that indistinguishability holds. We note that most of the known garbling schemes based on Yao and optimizations [Yao82, ZRE15, KS08] have simulators that comply with the above.

- HYB<sub>3,2</sub>: When the execution results in  $P_1$  evaluating GCs during  $\text{fair}_1$  and output  $y$ , the GC is created as  $(C'_2, d_2) \leftarrow \mathcal{S}_{\text{prv}}(1^\kappa, C, y, \mathbf{X}_2 = \{e_{2\alpha}^\alpha, e_{2(\ell+\alpha)}^\alpha, e_{2(2\ell+\alpha)}^\alpha, e_{2(3\ell+\alpha)}^\alpha\}_{\alpha \in [\ell]})$ . The commitment  $c_2$  is later equivocated to  $C'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, C'_2, t_2)$ . The commitment  $c_2^{dec}$  to the decoding information is created for a dummy value and later equivocated to  $d_2$  using  $o_{d_2}$  computed via  $o_{d_2} \leftarrow \text{Equiv}(c_2^{dec}, d_2, t_2)$ . The set of ciphertexts  $ct$  and  $z_1$  (if) generated use  $d_2$ .
- HYB<sub>4</sub>: Same as HYB<sub>2</sub>, except the following:
  - HYB<sub>4,1</sub>: When the execution results in  $P_1$  evaluating GCs during  $\text{fair}_1$  but results in abort,  $C_3$  is created as  $C'_3 \leftarrow \mathcal{S}_{\text{obv}}(1^\kappa, C, \mathbf{X}_3 =$

Fig. 16: Description of  $\mathcal{S}_{\text{fair}}$

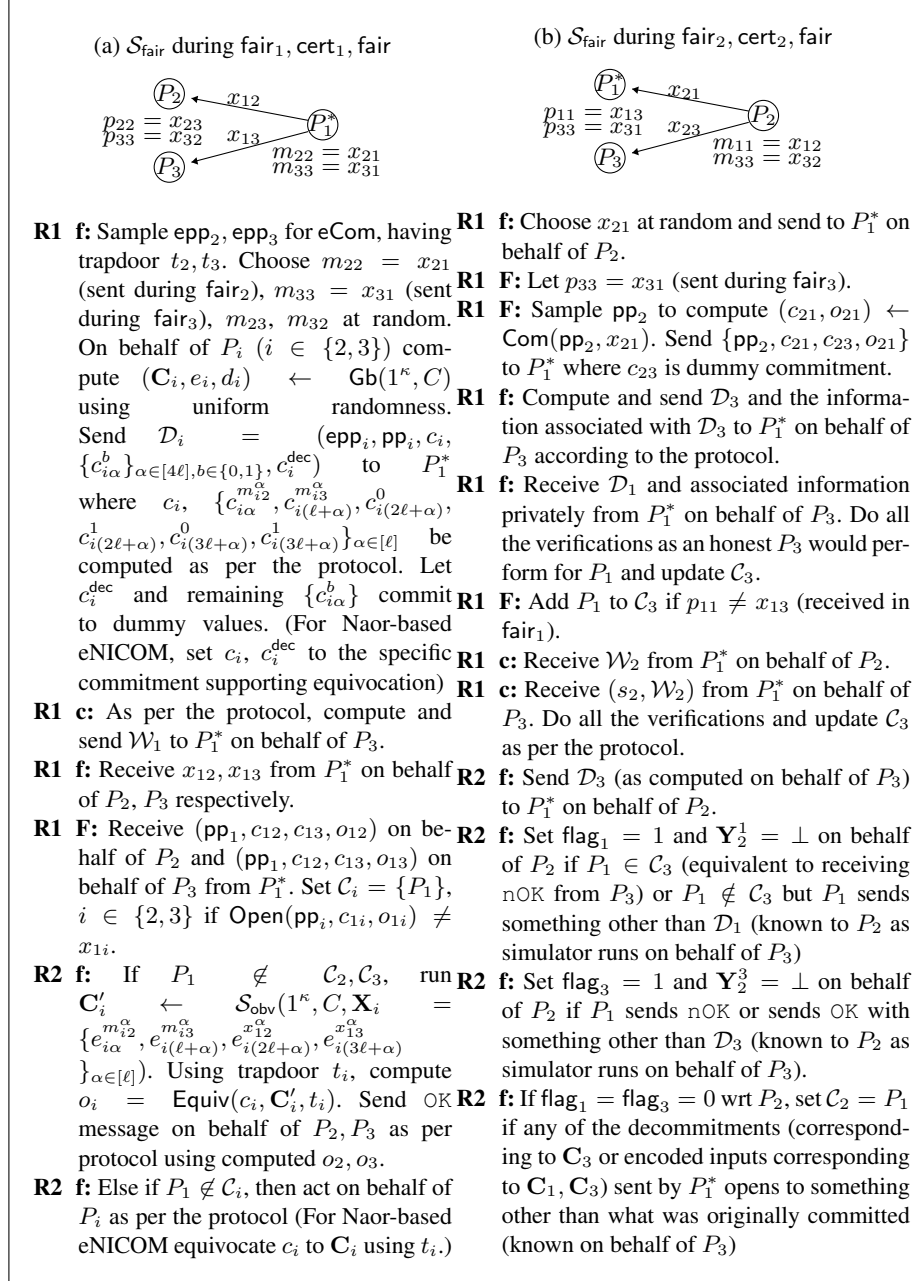


Fig. 17: Description of  $\mathcal{S}_{\text{fair}}$  (contd.)

- R2 F:** Set  $\text{flag}_1 = 1$  on behalf of both  $P_2, P_3$  if either  $P_1 \in \mathcal{C}_2$  or  $P_1 \in \mathcal{C}_3$  or  $\{\text{pp}_1, c_{12}, c_{13}\}$  received on behalf of  $P_2, P_3$  are not identical.
- R2 F:** Send ciphertext  $\text{ct}$  on dummy message on behalf of  $P_i$  if  $P_1 \notin \mathcal{C}_i$  ( $i \in \{2, 3\}$ ).
- R2 F:** If  $P_1 \notin \mathcal{C}_i$ ,  $\gamma_i = \{\mathcal{D}_1^1, \mathcal{D}_1^3, \mathcal{W}_2, \text{pp}_1, c_{12}, c_{13}\}$  received from  $P_1$  on behalf of  $P_i$  ( $i \in \{2, 3\}$ ).
- R2 c:** If  $P_1 \notin \mathcal{C}_2$ , send  $o_1, \mathcal{W}_1$  (same as computed on behalf of  $P_3$  in Round 1) and (opening of) encoding of  $\gamma_2$  to  $P_1$  on behalf of  $P_2$  as per the protocol.
- R2 c:** If  $P_1 \notin \mathcal{C}_3$ , send (opening of) encoding of  $\gamma_3$  to  $P_1$  on behalf of  $P_3$ .
- $\mathcal{S}_{\text{fair}}$  during Round 3:
- R3** Suppose  $\mathcal{C}_2 = \emptyset$ ,  $\text{flag}_1 = \text{flag}_3 = 0$  wrt  $P_2$ : If  $P_1$  sends encoded inputs corresponding to mismatched input bit across  $\mathbf{C}_1, \mathbf{C}_3$  during  $\text{fair}_2$  (known on behalf of  $P_3$ ), mark  $P_2$  as being in  $\text{st}_1$ . Invoke  $\mathcal{F}_{\text{fair}}$  with  $(\text{sid}, \text{Input}, x_1)$  to obtain  $y$  where  $x_1 = x_{12} \oplus x_{13}$ . Send  $(y, o_{13})$  to  $P_1$  on behalf of  $P_2$ . Similar steps are executed on behalf of  $P_3$  if  $\mathcal{C}_3 = \emptyset$ ,  $\text{flag}_1 = \text{flag}_2 = 0$ .
- R3** For every input bit of  $P_3$ , choose  $s$  bits uniformly at random, say  $b_1, \dots, b_s$ . Using key based on  $P_3$ 's consistent input  $b_\alpha$  ( $\alpha \in [s]$ ) used in  $\mathbf{C}_1, \mathbf{C}_3$  during  $\text{fair}_2$ , try to decrypt ciphertext  $\text{ct}_{1\alpha}^\beta$  for ( $\beta \in \{0, 1\}$ ) received from  $P_1$  in Round 2. If the decryption is successful and the openings retrieved are same as  $(o_{13}, o_{31})$ , mark  $P_2$  as being in  $\text{st}_1$  and do the following: invoke  $\mathcal{F}_{\text{fair}}$  with  $(\text{sid}, \text{Input}, x_1)$  to obtain  $y$  where  $x_1 = x_{12} \oplus x_{13}$ . Send  $(y, o_{13})$  to  $P_1$  on behalf of  $P_2$ . Similar steps are executed by the simulator on behalf of  $P_3$ .
- R3** If  $P_2$  or  $P_3$  is in  $\text{st}_2$ , let  $x_1 = x_{12} \oplus x_{13}$ . Invoke  $\mathcal{F}_{\text{fair}}$  with  $(\text{sid}, \text{Input}, x_1)$  to obtain  $y$ .
- R3** If  $P_2$  in  $\text{st}_2$ , to retrieve decoding information  $o_3^{\text{dec}}$ : Run  $(\mathbf{C}_3, d_3^1) \leftarrow \mathcal{S}_{\text{priv}}(1^\kappa, C, y, \mathbf{X} = \{e_{3\alpha}^{m_{12}^\alpha}, e_{3(\ell+\alpha)}^{m_{13}^\alpha}, e_{3(2\ell+\alpha)}^{x_{12}^\alpha}, e_{3(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . Equivocate the commitment on decoding information in  $\text{fair}_1$  ( $c_3^{\text{dec}}$ ) to get  $o_3^{\text{dec}} = \text{Equiv}(c_3^{\text{dec}}, d_3^1, t_3)$ . Send  $(\mathbf{Y}_2, \text{cert}_2, o_3^{\text{dec}})$  to  $P_1^*$  on behalf of  $P_2$ . Here  $\text{cert}_2$  is set as encoding of 1 on output wire of  $\mathbf{C}_2$  during  $\text{cert}_2$  and  $\mathbf{Y}_2$  is the encoding corresponding to output  $y$  of  $\mathbf{C}_1, \mathbf{C}_3$  during  $\text{fair}_2$ ; both of which are known as simulator acts on behalf of  $P_3$ .
- R3** Similar steps as above if  $P_3$  is in  $\text{st}_2$ .
- R3** Invoke  $\mathcal{F}_{\text{fair}}$  with  $(\text{sid}, \text{Input}, \text{abort})$  if neither  $P_2$  nor  $P_3$  belong to  $\{\text{st}_1, \text{st}_2\}$ .
- R3** Send dummy ciphertext  $z_1$  to  $P_1^*$  on behalf of  $P_i$ ,  $i \in \{2, 3\}$  if  $P_i$  in  $\text{st}_4$  with  $\text{flag}_1 = 1$ .

$\{e_{3\alpha}^{m_{32}^\alpha}, e_{3(\ell+\alpha)}^{m_{33}^\alpha}, e_{3(2\ell+\alpha)}^{x_{12}^\alpha}, e_{3(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]}$ . The commitment  $c_3$  is later equiv-

- ocated to  $C'_3$  using  $o_3$  computed via  $o_3 \leftarrow \text{Equiv}(c_3, C'_3, t_3)$ . The commitment to the decoding information is created for a dummy value.
- HYB<sub>4.2</sub>: When the execution results in  $P_1$  evaluating GCs during fair<sub>1</sub> and output  $y$ , the GC is created as  $(C'_3, d_3) \leftarrow \mathcal{S}_{\text{prv}}(1^\kappa, C, y, \mathbf{X}_3 = \{e_{3\alpha}^{m_{32}^\alpha}, e_{3(\ell+\alpha)}^{m_{33}^\alpha}, e_{3(2\ell+\alpha)}^{x_{12}^\alpha}, e_{3(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . The commitment  $c_3$  is later equiv-ocated to  $C'_3$  using  $o_3$  computed via  $o_3 \leftarrow \text{Equiv}(c_3, C'_3, t_3)$ . The commitment  $c_3^{\text{dec}}$  to the decoding information is created for a dummy value and later equiv-ocated to  $d_3$  using  $o_{d_3}$  computed via  $o_{d_3} \leftarrow \text{Equiv}(c_3^{\text{dec}}, d_3, t_3)$ . The set of ciphertexts ct and  $z_1$  (if) generated uses  $d_3$ .
  - HYB<sub>5</sub>: Same as HYB<sub>4</sub>, except that during fair<sub>2</sub>,  $C_2$  is set to  $P_1$  if  $P_2$  receives  $o_3$  that opens to a value other than the originally committed  $C_3$ .
  - HYB<sub>6</sub>: Same as HYB<sub>5</sub>, except that during fair<sub>3</sub>,  $C_3$  is set to  $P_1$  if  $P_3$  receives  $o_2$  that opens to a value other than the originally committed  $C_2$ .
  - HYB<sub>7</sub>: Same as HYB<sub>6</sub>, except that during fair<sub>2</sub>,  $C_2$  is set to  $P_1$  if  $P_2$  accepts any encoded input not consistent with  $C_1, C_3$ .
  - HYB<sub>8</sub>: Same as HYB<sub>7</sub>, except that during fair<sub>3</sub>,  $C_3$  is set to  $P_1$  if  $P_3$  accepts any encoded input not consistent with  $C_1, C_2$ .
  - HYB<sub>9</sub>: Same as HYB<sub>8</sub>, except that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{23}$  (corresponding to  $x_{23}$ ) sent by  $P_2$  during fair<sub>2</sub>, the commitment is replaced with commitment of dummy value.
  - HYB<sub>10</sub>: Same as HYB<sub>9</sub>, except that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{32}$  (corresponding to  $x_{32}$ ) sent by  $P_3$  during fair<sub>3</sub>, the commitment is replaced with commitment of dummy value.
  - HYB<sub>11</sub>: Same as HYB<sub>10</sub>, except that when the execution fair<sub>1</sub> does not result in  $P_1$  getting encoded inputs corresponding to mismatched input bit across the two garbled circuits corresponding to any garbler, the set of ct is replaced by encryption of a dummy message.
  - HYB<sub>12</sub>: Same as HYB<sub>11</sub>, except that during cert<sub>2</sub>,  $P_2$  (with flag<sub>1</sub> = 0) adds  $P_1$  to  $C_2$  if (opening of) encoded input sent by  $P_1$  corresponding to  $C_2$  is anything other than the opening of the originally committed encoded information corresponding to value  $\gamma = \{\mathcal{D}_2^1, \mathcal{D}_2^3, \mathcal{W}_3, (\text{pp}_2, c_{21}, c_{23})\}$  sent by  $P_2$  in Round 1.
  - HYB<sub>13</sub>: Same as HYB<sub>12</sub>, except that during cert<sub>3</sub>,  $P_3$  (with flag<sub>2</sub> = 0) adds  $P_1$  to  $C_3$  if (opening of) encoded input sent by  $P_1$  corresponding to  $C_3$  is anything other than the opening of the originally committed encoded information corresponding to value  $\gamma = \{\mathcal{D}_3^1, \mathcal{D}_3^2, \mathcal{W}_1, (\text{pp}_3, c_{31}, c_{32})\}$  sent by  $P_3$  in Round 1.
  - HYB<sub>14</sub>: Same as HYB<sub>13</sub>, except that during cert<sub>1</sub>, when  $P_1$ 's evaluation of  $C_1$  does not result in output 1,  $z_1$  (if) sent to  $P_1$  is replaced with encryption of dummy message.
  - HYB<sub>15</sub>: Same as HYB<sub>14</sub>, except that  $Y_2^1, Y_2^3$  is computed via  $\text{De}(Y_2^1, d_1) = y$ ,  $\text{De}(Y_2^3, d_3) = y$ , (where  $d_1, d_3$  correspond to decoding information of  $C_1, C_3$  during fair<sub>2</sub>) rather than  $Y_2^1 = \text{Ev}(C_1, \mathbf{X})$ ,  $Y_2^3 = \text{Ev}(C_3, \mathbf{X})$ .
  - HYB<sub>16</sub>: Same as HYB<sub>15</sub>, except that  $Y_3^1, Y_3^2$  is computed via  $\text{De}(Y_3^1, d_1) = y$ ,  $\text{De}(Y_3^2, d_2) = y$  (where  $d_1, d_2$  correspond to decoding information of  $C_1, C_2$  during fair<sub>3</sub>) rather than  $Y_3^1 = \text{Ev}(C_1, \mathbf{X})$ ,  $Y_3^2 = \text{Ev}(C_2, \mathbf{X})$ .
  - HYB<sub>17</sub>: Same as HYB<sub>16</sub>, except that during cert<sub>2</sub>, if  $P_2$  gets access to  $Y_2 \leftarrow (C_2, \mathbf{X})$  such that  $\text{sDe}(Y_2) = 1$ , cert<sub>2</sub> =  $Y_2$  is computed via  $\text{De}(Y_2, d_2) = 1$



- (where  $d_2$  corresponds to decoding information of  $\mathbf{C}_2$  during  $\text{cert}_2$ ) rather than  $\mathbf{Y}_2 = \text{Ev}(\mathbf{C}_2, \mathbf{X})$
- HYB<sub>18</sub>: Same as HYB<sub>17</sub>, except that during  $\text{cert}_3$ , if  $P_3$  gets access to  $\mathbf{Y}_3 \leftarrow (\mathbf{C}_3, \mathbf{X})$  such that  $\text{sDe}(\mathbf{Y}_3) = 1$ ,  $\text{cert}_3 = \mathbf{Y}_3$  is computed via  $\text{De}(\mathbf{Y}_3, d_3) = 1$  (where  $d_3$  corresponds to decoding information of  $\mathbf{C}_3$  during  $\text{cert}_3$ ) rather than  $\mathbf{Y}_3 = \text{Ev}(\mathbf{C}_3, \mathbf{X})$
  - HYB<sub>19</sub>: Same as HYB<sub>18</sub>, except that  $P_2$  sends  $(y, o_{13})$  to  $P_1$  if decryption of ct sent by  $P_1$  during  $\text{fair}_2$  is successful (and includes openings of  $x_{13}, x_{31}$  corresponding to original commitments) using  $P_3$ 's encoding corresponding to random input.
  - HYB<sub>20</sub>: Same as HYB<sub>19</sub>, except that  $P_3$  sends  $(y, o_{12})$  to  $P_1$  if decryption of ct sent by  $P_1$  during  $\text{fair}_3$  is successful (and includes openings of  $x_{12}, x_{21}$  corresponding to original commitments) using  $P_2$ 's encoding corresponding to random input.

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

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

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

$\text{HYB}_2 \stackrel{c}{\approx} \text{HYB}_{3.1}$ : The difference between the hybrids is in the way  $(\mathbf{C}_2, \mathbf{X})$  is generated when the execution results in abort. In  $\text{HYB}_2$ ,  $(\mathbf{C}_2, e, d) \leftarrow \text{Gb}(1^\kappa, C)$  is run, which gives  $(\mathbf{C}_2, \text{En}(x, e))$ . In  $\text{HYB}_{3.1}$ , it is generated as  $\mathbf{C}'_2 \leftarrow \mathcal{S}_{\text{obv}}(1^\kappa, C, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . The commitment to the garbled circuit is later equivocated to  $\mathbf{C}'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, \mathbf{C}'_2, t_2)$ . Additionally, the commitment to the decoding information is created for a dummy value in  $\text{HYB}_{3.1}$ . The indistinguishability follows via reduction to the obliviousness of the garbling scheme and the usual hiding property of commitment schemes which is implied by the hiding property of eCom.

$\text{HYB}_2 \stackrel{c}{\approx} \text{HYB}_{3.2}$ : The difference between the hybrids is in the way  $(\mathbf{C}_2, \mathbf{X}, d)$  is generated. In  $\text{HYB}_2$ ,  $(\mathbf{C}_2, e, d) \leftarrow \text{Gb}(1^\kappa, C)$  is run, which gives  $(\mathbf{C}_2, \text{En}(x, e), d)$ . In  $\text{HYB}_{3.2}$ , it is generated as  $(\mathbf{C}'_2, d_2^1) \leftarrow \mathcal{S}_{\text{prv}}(1^\kappa, C, y, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . The commitment to the garbled circuit is later equivocated to  $\mathbf{C}'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, \mathbf{C}'_2, t_2)$ . Additionally, the commitment to the decoding information is created for a dummy value and later equivocated to  $d_2^1$  using  $o_2^{\text{dec}}$  computed via  $o_2^{\text{dec}} \leftarrow \text{Equiv}(c_2^{\text{dec}}, d_2^1, t_2)$ . The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of eCom.

$\text{HYB}_3 \stackrel{c}{\approx} \text{HYB}_4$ : Similar argument as above with respect to  $\mathbf{C}_3$ .

$\text{HYB}_4 \stackrel{c}{\approx} \text{HYB}_5$ : The difference between the hybrids is that in  $\text{HYB}_4$ ,  $P_2$  sets  $\mathbf{C}_2 = P_1$  if the  $o_3$  sent by  $P_1$  in  $\text{fair}_2$  output  $\perp$  while in  $\text{HYB}_5$ ,  $P_2$  sets  $\mathbf{C}_2 = P_1$  if  $o_3$  sent by  $P_1$  in  $\text{fair}_2$  opens to any value other than  $\mathbf{C}_3$ . Since the commitment scheme eCom

is binding, in  $\text{HYB}_4$ ,  $P_1$  could have decommitted successfully to a different garbled circuit than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_5 \stackrel{c}{\approx} \text{HYB}_6$ : Similar argument as above with respect to  $P_3$  in  $\text{fair}_3$ .

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

$\text{HYB}_7 \stackrel{c}{\approx} \text{HYB}_8$ : Similar argument as above with respect to  $P_3$  in  $\text{fair}_3$ .

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

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

$\text{HYB}_{10} \stackrel{c}{\approx} \text{HYB}_{11}$ : The difference between the hybrids is that when the execution  $\text{fair}_1$  does not result in  $P_1$  getting encoded inputs corresponding to mismatched input bits of any garbler on two garbled circuits, in  $\text{HYB}_{10}$ , the set of ct is the encryption of a opening of input shares while in  $\text{HYB}_{11}$ , it is replaced with encryption of dummy message. Assuming the encryption key is unknown to  $P_1$  (holds except with negligible probability due to privacy of garbling scheme), indistinguishability follows from the security of the encryption scheme with special correctness.

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

$\text{HYB}_{12} \stackrel{c}{\approx} \text{HYB}_{13}$ : Similar argument as above with respect to  $P_3$  during  $\text{cert}_3$ .

$\text{HYB}_{13} \stackrel{c}{\approx} \text{HYB}_{14}$ : The difference between the hybrids is that in  $\text{HYB}_{12}$ ,  $z_1$  is set as encryption of the decoding information of  $\text{fair}_1$  while in  $\text{HYB}_{13}$ ,  $z_1$  is replaced with encryption of a dummy message when  $P_1$ 's evaluation of  $\mathbf{C}_1$  during  $\text{cert}_1$  does not lead to output 1. Assuming the encryption key is unknown to  $P_1$  (holds except with negligible probability due to authenticity of privacy-free garbling scheme used in  $\text{cert}_1$ ), indistinguishability follows from the security of the encryption scheme.

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

scheme, the equivalence of  $\mathbf{Y}_2^1, \mathbf{Y}_2^3$  computed via  $\text{Ev}(\mathbf{C}_1, \mathbf{X}), \text{Ev}(\mathbf{C}_3, \mathbf{X})$  or such that  $\text{De}(\mathbf{Y}_2^1, d_1) = y, \text{De}(\mathbf{Y}_2^3, d_3) = y$  holds.

$\text{HYB}_{15} \stackrel{c}{\approx} \text{HYB}_{16}$ : Similar argument as above with respect to  $\mathbf{Y}_3$  computed by  $P_3$  during  $\text{fair}_3$ .

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

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

$\text{HYB}_{18} \stackrel{c}{\approx} \text{HYB}_{19}$ : The difference between the hybrids is that in  $\text{HYB}_{18}$ ,  $P_2$  sends  $(y, o_{13})$  to  $P_1$  if decryption of  $\text{ct}$  sent by  $P_1$  during  $\text{fair}_2$  is successful using keys based on  $P_3$ 's encoding of actual input, whereas in  $\text{HYB}_{19}$ ,  $P_2$  sends  $(y, o_{13})$  to  $P_1$  if decryption of  $\text{ct}$  sent by  $P_1$  during  $\text{fair}_2$  is successful using keys based on  $P_3$ 's encoding of random input. The indistinguishability between the hybrids follows from the following claim: Consider single bit input for simplicity. For any two different inputs  $x$  and  $x'$  of  $P_3$ , the difference between the probability that  $P_2$  sends  $(y, o_{13})$  to  $P_1$  when  $P_3$ 's input is  $x$  and when  $P_3$ 's input is  $x'$  is at most  $2^{-s+1}$ . The argument can be divided into three cases (similar to [LP07]). **(1)** Suppose for some  $\alpha \in [s]$ ,  $P_1$  replaces both ciphertexts  $\text{ct}_{1\alpha}^0, \text{ct}_{1\alpha}^1$ : one based on consistent input 0 of  $P_3$  and other based on consistent input 1 of  $P_3$  (say,  $\text{sk}_\alpha^0 = \mathbf{X}_{1(s+\alpha)}^0 \oplus \mathbf{X}_{3(s+\alpha)}^0$  and  $\text{sk}_\alpha^1 = \mathbf{X}_{1(s+\alpha)}^1 \oplus \mathbf{X}_{3(s+\alpha)}^1$ ). In this case,  $P_2$  would be able to decrypt the ciphertext successfully regardless of  $P_3$ 's input with probability 1 and would send  $(y, o_{13})$  to  $P_2$ . **(2)** Suppose  $P_1$  replaces exactly one of the two ciphertexts with consistent input corresponding to  $1 \leq j < s$ . Since the values assigned (in encoding) by  $P_3$  to any proper subset of the  $s$  bits are independent of  $P_3$ 's actual input,  $P_2$  would be able to decrypt the ciphertext successfully with probability  $1 - 2^{-j}$  regardless of the actual value of its original input. **(3)** Suppose  $P_1$  replaces one ciphertext based on consistent input for each of the  $\alpha \in [s]$  (say all based on consistent value '1'). Then if  $x$  had encoding with any one such value ('1' in the example), the ciphertext would be decrypted successfully with probability 1, whereas decryption would be successful with probability  $1 - 2^{-s+1}$  if  $x'$  had the other value (in the example,  $P_2$  will be unable to decrypt if  $x' = 0$  and the encoding of  $x' = 0$  was chosen as  $x'_\alpha = 0$  for all  $\alpha \in [s]$  (where  $x' = \bigoplus_{\alpha=1}^s x_\alpha$ ) which occurs with probability  $2^{-s+1}$ ).

$\text{HYB}_{19} \stackrel{c}{\approx} \text{HYB}_{20}$ : Similar argument as above with respect to  $\text{ct}$  received by  $P_3$  during  $\text{fair}_3$ .

## E Proof of Security for Protocol ua

In this section, we present the proof of security of  $\text{ua}$  relative to the ideal functionality for unanimous abort (Figure 11) shown in Appendix B. For clarity, we assume without loss of generality that  $P_1$  is corrupt (denoted as  $P_1^*$ ) and describe the simulator  $\mathcal{S}_{\text{ua}}$ . Since the roles of the parties are symmetric in  $\text{ua}$ , similar proof would hold in case of

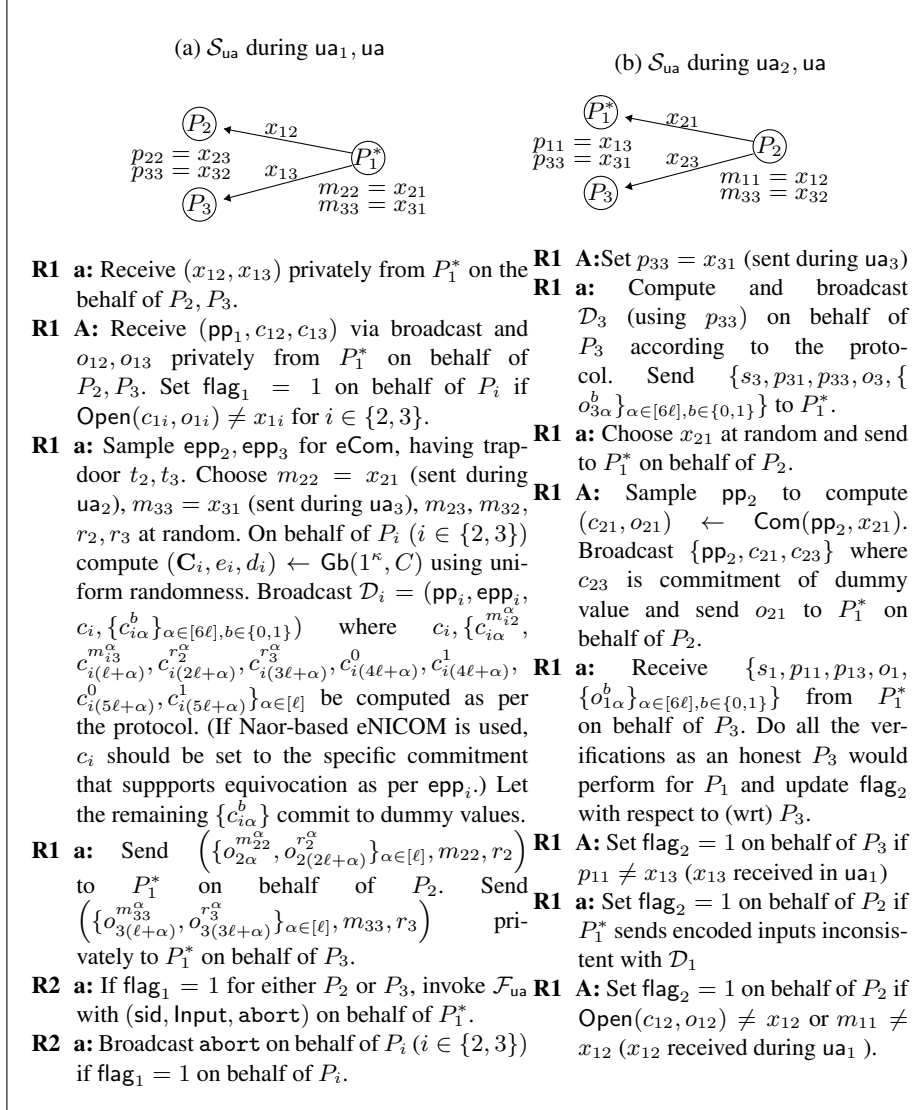
corrupt  $P_2, P_3$  as well. The simulator plays the role of the honest parties  $P_2, P_3$  and simulates each step of the protocol ua.

We divide the description of  $\mathcal{S}_{ua}$  as follows: We describe  $\mathcal{S}_{ua}$  during  $ua_1$  where corrupt  $P_1^*$  is the evaluator and during  $ua_2$  when corrupt  $P_1^*$  acts as a garbler. The steps corresponding to  $ua_3$ , would follow symmetrically from that described corresponding to  $ua_2$ . The simulator  $\mathcal{S}_{ua}$  appears in Figure 18-19 with **R1/R2** indicating simulation for round 1 and 2 respectively and **a/A** denoting the steps corresponding to subprotocol  $ua_i, ua$  respectively. When simulating  $ua_1$ , the commitments for GC and encoding information need to be simulated and sent in Round 1 itself, while the privacy simulator  $\mathcal{S}_{prv}$  can only be invoked on noting the adversary's behaviour in Round 1 that decides what input it commits and whether it obtains output or  $\perp$ . Using equivocality of the commitment of GC, we can equivocate the GC as returned by the simulator. But since commitments on the encoding information are committing and the simulator didn't have access to  $\mathbf{X}$  during simulation of Round 1, the encoded input  $\mathbf{X}$  returned by  $\mathcal{S}_{prv}$  cannot be explained. So we use a slightly modified version of  $\mathcal{S}_{prv}$  which takes an encoded input (correspond to what will be opened to corrupt  $P_1$ ) as parameter and returns just the fake GC compatible with it. Yao's privacy simulator can be made to work as above for any encoded input and the indistinguishability will hold with respect to the fake GC and given encoded input.

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

- HYB<sub>0</sub>: Same as  $\text{REAL}_{ua, \mathcal{A}}$ .
- HYB<sub>1</sub>: Same as HYB<sub>0</sub>, except that  $P_2, P_3$  in  $ua_1$  use uniform randomness rather than pseudo-randomness for the garbled circuit construction.
- HYB<sub>2</sub>: Same as HYB<sub>1</sub>, except that some of the commitments of encoded inputs which will not be sent to  $P_1$  during  $ua_1$  are replaced with commitment on dummy values. Specifically, these are corresponding to indices not equal to  $m_{22}, m_{23}, r_2, r_3, z_2, z_3$  for  $\mathbf{C}_2$  and not equal to  $m_{32}, m_{33}, r_2, r_3, z_2, z_3$  for  $\mathbf{C}_3$ .
- HYB<sub>3</sub> : Same as HYB<sub>2</sub>, except that when the execution results in  $P_1$  evaluating GCs during  $ua_1$ , the GC  $\mathbf{C}_2$  is created as  $(\mathbf{C}'_2, d_2) \leftarrow \mathcal{S}_{prv}(1^\kappa, C, y, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{r_2^\alpha}, e_{2(3\ell+\alpha)}^{r_3^\alpha}, e_{2(4\ell+\alpha)}^{z_2^\alpha}, e_{2(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha \in [\ell]})$ . The commitment  $c_2$  is later equivocated to  $\mathbf{C}'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, \mathbf{C}'_2, t_2)$ . The set of ciphertexts ct generated uses  $d_2$  in their keys.
- HYB<sub>4</sub> : Same as HYB<sub>3</sub>, except that when the execution results in  $P_1$  evaluating GCs during  $ua_1$ , the GC  $\mathbf{C}_3$  is created as  $(\mathbf{C}'_3, d_3) \leftarrow \mathcal{S}_{prv}(1^\kappa, C, y, \mathbf{X}_3 = \{e_{3\alpha}^{m_{32}^\alpha}, e_{3(\ell+\alpha)}^{m_{33}^\alpha}, e_{3(2\ell+\alpha)}^{r_2^\alpha}, e_{3(3\ell+\alpha)}^{r_3^\alpha}, e_{3(4\ell+\alpha)}^{z_2^\alpha}, e_{3(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha \in [\ell]})$ . The commitment  $c_3$  is later equivocated to  $\mathbf{C}'_3$  using  $o_3$  computed via  $o_3 \leftarrow \text{Equiv}(c_3, \mathbf{C}'_3, t_3)$ . The set of ciphertexts ct generated uses  $d_3$  in their keys.
- HYB<sub>5</sub>: Same as HYB<sub>4</sub>, except that during  $ua_2$ ,  $\text{flag}_2$  is set to 1 if  $\mathcal{W}_1$  broadcast by  $P_1$  has anything other than (opening of) encoded input corresponding to  $z_1$  in  $\mathbf{C}_1$ .
- HYB<sub>6</sub>: Same as HYB<sub>5</sub>, except that during  $ua_3$ ,  $\text{flag}_3$  is set to 1 if  $\mathcal{W}_1$  broadcast by  $P_1$  has anything other than (opening of) encoded input corresponding to  $z_1$  in  $\mathbf{C}_1$ .
- HYB<sub>7</sub>: Same as HYB<sub>6</sub>, except that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{23}$  (corresponding to  $x_{23}$ ) broadcast by  $P_2$  during  $ua_2$ , the commitment is replaced with commitment of dummy value.

Fig. 18: Description of  $\mathcal{S}_{ua}$



- HYB<sub>8</sub>: Same as HYB<sub>7</sub>, except that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{32}$  (corresponding to  $x_{32}$ ) broadcast by  $P_3$  during  $ua_3$ , the commitment is replaced with commitment of dummy value.
- HYB<sub>9</sub>: Same as HYB<sub>8</sub>, except that when the execution  $ua_1$  does not result in  $P_1$  getting conflicting output on two garbled circuits, the set of ct is replaced by encryption of a dummy message.

Fig. 19: Description of  $\mathcal{S}_{ua}$

(a) $\mathcal{S}_{ua}$ during $ua_1, ua$ (Contd.)	(b) $\mathcal{S}_{ua}$ during $ua_2, ua$ (Contd.)
<p><b>R2 a:</b> If <math>\text{flag}_1 = 0</math> wrt <math>P_i</math> for exactly one <math>i \in \{2, 3\}</math>, then act on behalf of <math>P_i</math> as per the protocol opening the garbled circuit (equivocate <math>c_i</math> to <math>\mathbf{C}_i</math> in case of Naor-based eNICOM) and encoded input as per <math>m_{23}</math> or <math>m_{32}</math> accordingly chosen as above. Broadcast <math>\mathcal{W}_i</math> using <math>z_i = r_i \oplus x_{1i}</math> as per the protocol.</p> <p><b>R2 a:</b> If <math>\text{flag}_1 = 0</math> for both <math>P_2</math> and <math>P_3</math>, invoke <math>\mathcal{F}_{ua}</math> with <math>(\text{sid}, \text{Input}, x_1)</math> on behalf of <math>P_1^*</math> to obtain output <math>y</math>, where <math>x_1 = x_{12} \oplus x_{13}</math>. Let <math>z_2 = r_2 \oplus x_{12}</math> and <math>z_3 = r_3 \oplus x_{13}</math>. For <math>(i \in \{2, 3\})</math>, run <math>(\mathbf{C}'_i, \mathbf{X}_i, d_i) \leftarrow \mathcal{S}_{\text{priv}}(1^\kappa, C, y, \{e_{i\alpha}^{m_{i2}^\alpha}, e_{i(\ell+\alpha)}^{m_{i3}^\alpha}, e_{i(2\ell+\alpha)}^{r_2^\alpha}, e_{i(3\ell+\alpha)}^{r_3^\alpha}, e_{i(4\ell+\alpha)}^{z_2^\alpha}, e_{i(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha \in [\ell]}).</math> Using trapdoor <math>t_i</math>, compute <math>o_i = \text{Equiv}(c_i, \mathbf{C}'_i, t_i)</math>. Send OK message privately to <math>P_1^*</math> on behalf of <math>P_2, P_3</math> as per the protocol using computed <math>o_2, o_3</math>. Broadcast <math>\mathcal{W}_2, \mathcal{W}_3</math> on behalf of <math>P_2, P_3</math> as per protocol.</p> <p><b>R2 A:</b> If <math>\text{flag}_1 \neq 1</math> wrt <math>P_i</math>, send set of ct on behalf of <math>P_i</math> (<math>i \in \{2, 3\}</math>) using a dummy message.</p> <p><b>R2 a:</b> Set <math>\text{flag}_1 = 1</math> on behalf of both <math>P_2, P_3</math> if either (a) abort was sent or received via broadcast in Round 2 (b) <math>P_1</math> broadcasts <math>z_2 \neq x_{12} \oplus r_2</math> or <math>z_3 \neq x_{13} \oplus r_3</math></p>	<p><b>R2 a:</b> On behalf of <math>P_3</math>: If <math>\text{flag}_2 = 0</math> wrt <math>P_3</math>, choose random <math>z_3</math> and broadcast <math>\mathcal{W}_3</math> as per the protocol. Else broadcast abort.</p> <p><b>R2 a:</b> On behalf of <math>P_2</math>: If <math>\text{flag}_2 = 0</math> wrt <math>P_2</math>, broadcast <math>z_1, z_3</math> where <math>z_1</math> is computed as per the protocol as <math>z_1 = x_{21} \oplus r_1</math>, where <math>x_{21}</math> sent to <math>P_1^*</math> in Round 1 and <math>r_1</math> received from <math>P_1^*</math>. <math>z_3</math> is either same as chosen above (if <math>\text{flag}_2 = 0</math> wrt <math>P_3</math>) or random (if <math>\text{flag}_2 = 1</math> wrt <math>P_3</math>). Else broadcast abort.</p> <p><b>R2 a:</b> Set <math>\text{flag}_2 = 0</math> on behalf of both <math>P_2, P_3</math> if (a) abort was sent or received via broadcast in Round 2 (b) <math>P_1^*</math> broadcasts anything other than <math>(z_1, o_{1(4\ell+\alpha)}^{z_1^\alpha})</math> (<math>o_{1(4\ell+\alpha)}^{z_1^\alpha}</math> known on behalf of <math>P_3</math>) where <math>z_1 = x_{21} \oplus r_1</math> (<math>r_1, x_{21}</math> known to <math>P_2</math>)</p>
<p><math>\mathcal{S}_{ua}</math> after <math>ua_1, ua_2, ua_3</math>: If <math>\text{flag}_i = 1</math> (on behalf of both <math>P_2, P_3</math>) for any <math>i \in [3]</math>, invoke <math>\mathcal{F}_{ua}</math> with abort on behalf of <math>P_1^*</math>. Else invoke <math>\mathcal{F}_{ua}</math> with continue on behalf of <math>P_1^*</math>.</p>	

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

$\text{HYB}_0 \stackrel{c}{\approx} \text{HYB}_1$ : The difference between the hybrids is that  $P_2, P_3$  in  $ua_1$  use uniform randomness in  $\text{HYB}_1$  rather than pseudorandomness as in  $\text{HYB}_0$ . The indistinguishability follows via reduction to the security of the PRG  $G$ .

$\text{HYB}_1 \stackrel{c}{\approx} \text{HYB}_2$ : The difference between the hybrids is some of the commitments of encoded inputs which will not be sent to  $P_1$  during  $ua_1$  are replaced with commitment on dummy messages. The indistinguishability follows from the hiding property of NICOM.

$\text{HYB}_2 \stackrel{c}{\approx} \text{HYB}_3$ : The difference between the hybrids is in the way  $(\mathbf{C}_2, \mathbf{X}, d_2)$  is generated. In  $\text{HYB}_2$ ,  $(\mathbf{C}_2, e_2, d_2) \leftarrow \text{Gb}(1^\kappa, C)$  is run, which gives

$(\mathbf{C}_2, \text{En}(x, e), d_2)$ . In  $\text{HYB}_3$ , it is generated as  $(\mathbf{C}'_2, d_2) \leftarrow \mathcal{S}_{\text{prv}}(1^\kappa, C, y, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{r_2^\alpha}, e_{2(3\ell+\alpha)}^{r_3^\alpha}, e_{2(4\ell+\alpha)}^{z_2^\alpha}, e_{2(5\ell+\alpha)}^{z_3^\alpha}\}_{\alpha \in [\ell]})$ . The commitment to the garbled circuit is later equivocated to  $\mathbf{C}'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, \mathbf{C}'_2, t_2)$ . The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of  $\text{eCom}$ .

$\text{HYB}_3 \stackrel{c}{\approx} \text{HYB}_4$ : Similar argument as above with respect to  $\mathbf{C}_3$ .

$\text{HYB}_4 \stackrel{c}{\approx} \text{HYB}_5$ : The difference between the hybrids is that in  $\text{HYB}_4$ ,  $\text{flag}_2$  is set to 1 if  $\mathcal{W}_1$  broadcast by  $P_1$  during  $\text{ua}_2$  has (opening of) encoded input that is inconsistent with commitment corresponding to  $z_1$  in  $\mathcal{D}_1$ , while in  $\text{HYB}_5$ ,  $\text{flag}_2$  is set to 1 if  $\mathcal{W}_1$  broadcast by  $P_1$  has (opening of) encoded input anything other than encoding of  $z_1$  corresponding to  $\mathbf{C}_1$ . It follows from the binding property of  $\text{NICOM}$  that  $P_1$  could have sent an encoded input not consistent with  $\mathbf{C}_1$  but consistent with  $\mathcal{D}_1$ , only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_5 \stackrel{c}{\approx} \text{HYB}_6$ : Similar argument as above with respect to  $\mathcal{W}_1$  broadcast by  $P_1$  during  $\text{ua}_3$ .

$\text{HYB}_6 \stackrel{c}{\approx} \text{HYB}_7$ : The difference between the hybrids is that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{23}$  (corresponding to  $x_{23}$ ) broadcast by  $P_2$  during  $\text{ua}_2$ ,  $c_{23}$  corresponds to the actual input share  $x_{23}$  in  $\text{HYB}_8$  while it corresponds to dummy value in  $\text{HYB}_9$ . The indistinguishability follows from the hiding property of  $\text{NICOM Com}$ .

$\text{HYB}_7 \stackrel{c}{\approx} \text{HYB}_8$ : Similar argument as above with respect to commitment  $c_{32}$  broadcast by  $P_3$  during  $\text{ua}_3$ .

$\text{HYB}_8 \stackrel{c}{\approx} \text{HYB}_9$ : The difference between the hybrids is that when the execution  $\text{ua}_1$  does not result in  $P_1$  getting conflicting output on two garbled circuits, in  $\text{HYB}_8$ , the set of  $\text{ct}$  is the encryption of opening of shares of input while in  $\text{HYB}_9$ , it is replaced with encryption of dummy message. Assuming the encryption key is unknown to  $P_1$  (holds except with negligible probability due to authenticity), indistinguishability follows from the CPA security of the encryption scheme.

## F Proof of Security for Protocol $\text{god}$

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

Similar to  $\mathcal{S}_{\text{ua}}$ , we divide the description of  $\mathcal{S}_{\text{god}}$  as follows: We describe  $\mathcal{S}_{\text{god}}$  during  $\text{god}_1$  where corrupt  $P_1^*$  is the evaluator and during  $\text{god}_2$  when corrupt  $P_1^*$  acts as a garbler. The steps corresponding to  $\text{god}_3$ , would follow symmetrically from that described corresponding to  $\text{god}_2$ . We then describe the steps of the simulator  $\mathcal{S}_{\text{god}}$  corresponding to the third round. In the protocol  $\text{god}$ , the behavior of corrupt  $P_1$  in Round 1, 2 determines his committed input. Hence, the privacy simulator can only be invoked earliest after the simulation of the first round. Similar to  $\mathcal{S}_{\text{ua}}$ , since the commitments on

encoding information is sent in the first round itself, we use a modified version of the privacy simulator of the garbling scheme which additionally takes an encoded input as parameter (see Section E). The simulator  $\mathcal{S}_{\text{god}}$  appears in Figure 20-21 with **R1/R2/R3** indicating simulation for round 1, 2 and 3 and **g/G** denoting the steps corresponding to subprotocol  $\text{god}_i$ ,  $\text{god}$  respectively.

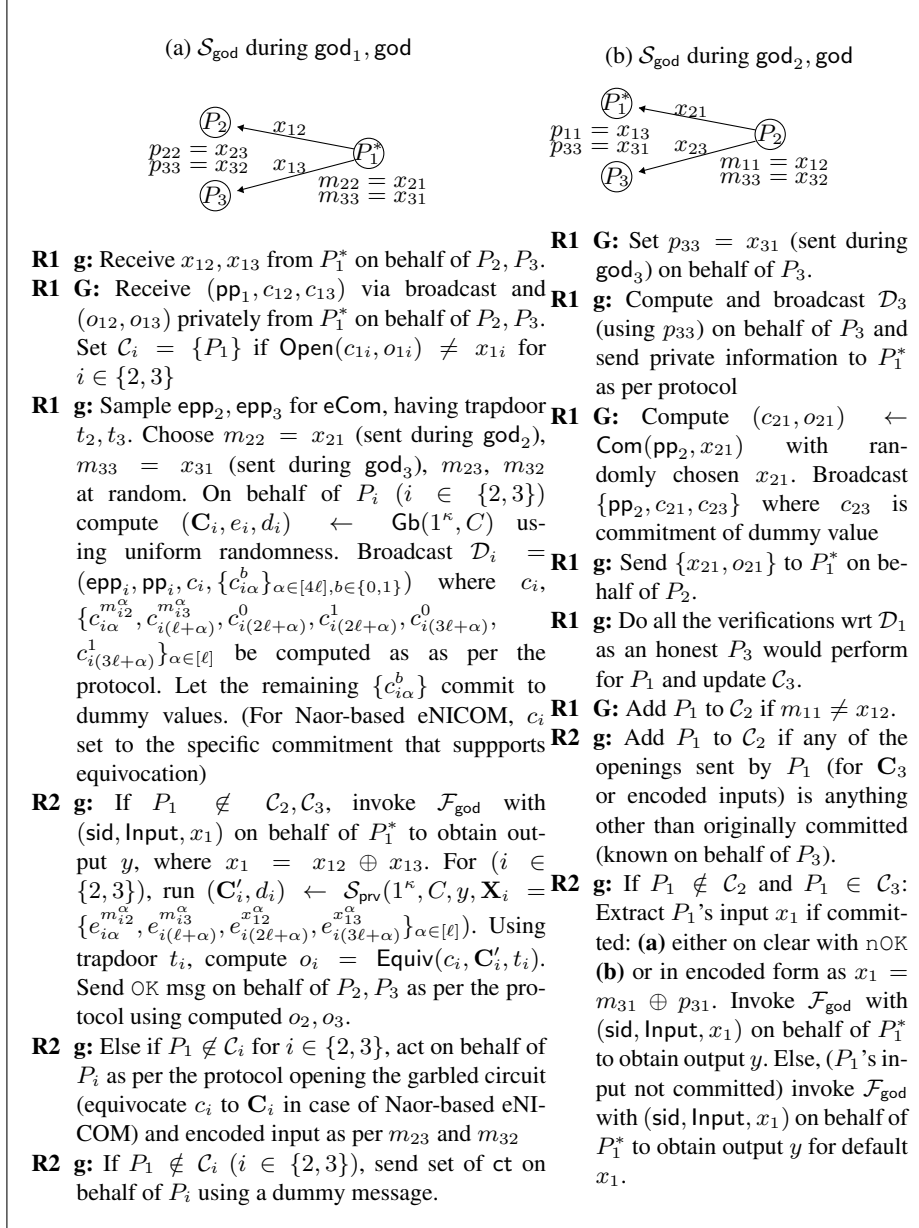
We now argue that  $\text{IDEAL}_{\mathcal{F}_{\text{god}}, \mathcal{S}_{\text{god}}} \stackrel{c}{\approx} \text{REAL}_{\text{god}, \mathcal{A}}$ , when  $\mathcal{A}$  corrupts  $P_1$ . The views are shown to be indistinguishable via a series of intermediate hybrids.

- $\text{HYB}_0$ : Same as  $\text{REAL}_{\text{god}, \mathcal{A}}$ .
- $\text{HYB}_1$ : Same as  $\text{HYB}_0$ , except that  $P_2, P_3$  in  $\text{god}_1$  use uniform randomness rather than pseudo-randomness for the garbled circuit construction.
- $\text{HYB}_2$ : Same as  $\text{HYB}_1$ , except that some of the commitments that will not be opened by  $P_1$  during  $\text{god}_1$  are replaced with commitment on dummy values. Specifically, these are corresponding to indices not equal to  $m_{22}, m_{23}, x_{12}, x_{13}$  for  $\mathbf{C}_2$  and not equal to  $m_{32}, m_{33}, x_{12}, x_{13}$  for  $\mathbf{C}_3$ .
- $\text{HYB}_3$  : Same as  $\text{HYB}_2$ , except that when the execution results in  $P_1$  evaluating GCs during  $\text{god}_1$ , the GC  $\mathbf{C}_2$  is created as  $(\mathbf{C}'_2, d_2) \leftarrow \mathcal{S}_{\text{prv}}(1^\kappa, C, y, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . The commitment  $c_2$  is later equivocated to  $\mathbf{C}'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, \mathbf{C}'_2, t_2)$ . The set of ciphertexts  $\text{ct}$  generated uses  $d_2$  in their keys.
- $\text{HYB}_4$  : Same as  $\text{HYB}_3$ , except that when the execution results in  $P_1$  evaluating GCs during  $\text{god}_1$ , the GC  $\mathbf{C}_3$  is created as  $(\mathbf{C}'_3, d_3) \leftarrow \mathcal{S}_{\text{prv}}(1^\kappa, C, y, \mathbf{X}_3 = \{e_{3\alpha}^{m_{32}^\alpha}, e_{3(\ell+\alpha)}^{m_{33}^\alpha}, e_{3(2\ell+\alpha)}^{x_{12}^\alpha}, e_{3(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . The commitment  $c_3$  is later equivocated to  $\mathbf{C}'_3$  using  $o_3$  computed via  $o_3 \leftarrow \text{Equiv}(c_3, \mathbf{C}'_3, t_3)$ . The set of ciphertexts  $\text{ct}$  generated uses  $d_3$  in their keys.
- $\text{HYB}_5$ : Same as  $\text{HYB}_4$ , except that during  $\text{god}_2$ ,  $\mathbf{C}_2$  is set to  $P_1$  if  $P_2$  receives  $o_3$  that opens to a value other than the originally committed  $\mathbf{C}_3$ .
- $\text{HYB}_6$ : Same as  $\text{HYB}_5$ , except that during  $\text{god}_3$ ,  $\mathbf{C}_3$  is set to  $P_1$  if  $P_3$  receives  $o_2$  that opens to a value other than the originally committed  $\mathbf{C}_2$ .
- $\text{HYB}_7$ : Same as  $\text{HYB}_6$ , except that during  $\text{god}_2$ ,  $\mathbf{C}_2$  is set to  $P_1$  if  $P_2$  accepts any encoded input not consistent with  $\mathbf{C}_1, \mathbf{C}_3$
- $\text{HYB}_8$ : Same as  $\text{HYB}_7$ , except that during  $\text{god}_3$ ,  $\mathbf{C}_3$  is set to  $P_1$  if  $P_3$  accepts any encoded input not consistent with  $\mathbf{C}_1, \mathbf{C}_2$
- $\text{HYB}_9$ : Same as  $\text{HYB}_8$ , except that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{23}$  (corresponding to  $x_{23}$ ) broadcast by  $P_2$  during  $\text{god}_2$ , the commitment is replaced with commitment of dummy value.
- $\text{HYB}_{10}$ : Same as  $\text{HYB}_9$ , except that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{32}$  (corresponding to  $x_{32}$ ) broadcast by  $P_3$  during  $\text{god}_3$ , the commitment is replaced with commitment of dummy value.
- $\text{HYB}_{11}$ : Same as  $\text{HYB}_{10}$ , except that when the execution  $\text{god}_1$  does not result in  $P_1$  getting conflicting output on two garbled circuits, the set of  $\text{ct}$  is replaced by encryption of a dummy message.

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



Fig. 20: Description of  $\mathcal{S}_{\text{god}}$



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

Fig. 21: Description of  $\mathcal{S}_{\text{god}}$  (contd.)

$\mathcal{S}_{\text{god}}$  during **R3**: If  $P_1 \in \mathcal{C}_2, \mathcal{C}_3$  at the end of round 1, invoke  $\mathcal{F}_{\text{god}}$  with  $(\text{sid}, \text{Input}, x_1)$  on behalf of  $P_1^*$  to obtain  $y$  for a default  $x_1$ . Send  $y$  to  $P_1^*$  on behalf of both  $P_2$  and  $P_3$  if  $P_1 \in \mathcal{C}_2, \mathcal{C}_3$  in the end of round one. Send  $y$  to  $P_1^*$  on behalf of only  $P_2$  ( $P_3$ ) if  $P_1 \in \mathcal{C}_3$  ( $\mathcal{C}_2$ ) in the end of round one.

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

$\text{HYB}_2 \stackrel{c}{\approx} \text{HYB}_3$ : The difference between the hybrids is in the way  $(\mathbf{C}_2, \mathbf{X}, d_2)$  is generated. In  $\text{HYB}_2$ ,  $(\mathbf{C}_2, e_2, d_2) \leftarrow \text{Gb}(1^\kappa, C)$  is run, which gives  $(\mathbf{C}_2, \text{En}(x, e), d_2)$ . In  $\text{HYB}_3$ , it is generated as  $(\mathbf{C}'_2, d_2) \leftarrow \mathcal{S}_{\text{priv}}(1^\kappa, C, y, \mathbf{X}_2 = \{e_{2\alpha}^{m_{22}^\alpha}, e_{2(\ell+\alpha)}^{m_{23}^\alpha}, e_{2(2\ell+\alpha)}^{x_{12}^\alpha}, e_{2(3\ell+\alpha)}^{x_{13}^\alpha}\}_{\alpha \in [\ell]})$ . The commitment to the garbled circuit is later equivocated to  $\mathbf{C}'_2$  using  $o_2$  computed via  $o_2 \leftarrow \text{Equiv}(c_2, \mathbf{C}'_2, t_2)$ . The indistinguishability follows via reduction to the privacy of the garbling scheme and the hiding property of eCom.

$\text{HYB}_3 \stackrel{c}{\approx} \text{HYB}_4$ : Similar argument as above with respect to  $\mathbf{C}_3$ .

$\text{HYB}_4 \stackrel{c}{\approx} \text{HYB}_5$ : The difference between the hybrids is that in  $\text{HYB}_4$ ,  $P_2$  sets  $\mathcal{C}_2 = P_1$  if the  $o_3$  sent by  $P_1$  in  $\text{god}_2$  output  $\perp$  while in  $\text{HYB}_5$ ,  $P_2$  sets  $\mathcal{C}_2 = P_1$  if  $o_3$  sent by  $P_1$  in  $\text{god}_2$  opens to any value other than  $\mathcal{C}_3$ . Since the commitment scheme eCom is binding and epp was chosen uniformly at random by  $P_3$ , in  $\text{HYB}_4$ ,  $P_1$  could have decommitted successfully to a different garbled circuit than what was originally committed, only with negligible probability. Therefore, the hybrids are indistinguishable.

$\text{HYB}_5 \stackrel{c}{\approx} \text{HYB}_6$ : Similar argument as above with respect to  $P_3$  in  $\text{god}_3$ .

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

$\text{HYB}_7 \stackrel{c}{\approx} \text{HYB}_8$ : Similar argument as above with respect to  $P_3$  in  $\text{god}_3$ .

$\text{HYB}_8 \stackrel{c}{\approx} \text{HYB}_9$ : The difference between the hybrids is that when the execution does not result in  $P_1$  getting access to the opening of commitment  $c_{23}$  (corresponding to  $x_{23}$ ) broadcast by  $P_2$  during  $\text{god}_2$ ,  $c_{23}$  corresponds to the actual input share  $x_{23}$  in  $\text{HYB}_8$  while it corresponds to dummy value in  $\text{HYB}_9$ . The indistinguishability follows from the hiding property of Com.

$\text{HYB}_9 \stackrel{c}{\approx} \text{HYB}_{10}$ : Similar argument as above with respect to commitment  $c_{32}$  broadcast by  $P_3$  during  $\text{god}_3$ .

$\text{HYB}_{10} \stackrel{c}{\approx} \text{HYB}_{11}$ : The difference between the hybrids is that when the execution  $\text{god}_1$  does not result in  $P_1$  getting conflicting output on two garbled circuits, in  $\text{HYB}_{10}$ , the set

of  $ct$  is the encryption of an input and a share of input while in  $\text{HYB}_{11}$ , it is replaced with encryption of dummy message. Assuming the encryption key is unknown to  $P_1$  (holds except with negligible probability due to authenticity), indistinguishability follows from the CPA security of the encryption scheme.

## G Authenticated Conditional Disclosure of Secret

The subprotocol  $\text{cert}_i$  (Figure 2) used in our protocol  $\text{fair}$  is reminiscent of the notion of ‘conditional disclosure of secrets (CDS)’ which was first introduced in [GIKM00]. Informally, the problem of conditional disclosure of secrets involves two parties Alice and Bob, who hold inputs  $x$  and  $y$  respectively and wish to release a common secret  $s$  to Carol (who knows both  $x$  and  $y$ ) if only if the input  $(x, y)$  satisfies some predefined predicate  $f$ . The model allows Alice and Bob to have access to shared random string (hidden from Carol) and the only communication allowed is a single unidirectional message sent from each player (Alice and Bob) to Carol. Traditionally, CDS involves two properties, namely *correctness* (if  $f(x, y)$  is true, then Carol is always able to reconstruct  $s$  from her input and the messages she receives) and *privacy* (if  $f(x, y)$  is false, Carol obtains no information about the secret  $s$ ). Formally,

**Definition 8 (Conditional Disclosure of Secret).** [AARV17] *Let  $f : \mathcal{X} \times \mathcal{Y} \rightarrow \{0, 1\}$  be a predicate. Let  $F_1 : \mathcal{X} \times \mathcal{S} \times \mathcal{R} \rightarrow \mathcal{T}_1$  and  $F_2 : \mathcal{Y} \times \mathcal{S} \times \mathcal{R} \rightarrow \mathcal{T}_2$  be deterministic encoding algorithms, where  $\mathcal{S}$  is the secret domain. Then, the pair  $(F_1, F_2)$  is a CDS scheme for  $f$  if the function  $F(x, y, s, r) = (F_1(x, s, r), F_2(y, s, r))$  that corresponds to the joint computation of  $F_1$  and  $F_2$  on a common  $s$  and  $r$ , satisfies the following:*

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

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

**Definition 9 (Authenticated Conditional Disclosure of Secret).** *Let  $A, B$  denote two parties holding inputs  $x \in \mathcal{X}$  and  $y \in \mathcal{Y}$  respectively and having access to common secret  $s \in \mathcal{S}$  and  $C$  denote an external party. We assume a PPT adversary  $\mathcal{A}$  who can actively corrupt at most 1 party among  $A, B$  and  $C$ . An authenticated CDS protocol is secure against  $\mathcal{A}$  if the following properties hold:*

- $\delta$ -correctness holds for honest  $A, B$ , and  $C$  where  $\delta = \text{negl}(\kappa)$ .
- $\epsilon$ -privacy holds against  $\mathcal{A}$  corrupt  $C$ , where  $\epsilon = \text{negl}(\kappa)$ .

- *Authenticity: For 1-input  $(x, y)$  of  $f$  and any secret  $s$ , Dec may result in  $\perp$  when either  $A$  or  $B$  is corrupt, in which case  $C$  either identifies a corrupt party or a pair of parties in conflict that includes the corrupt party.*

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