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) [37, 19, 67], 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 [37, 27, 12, 28, 31, 16, 2] and MPC with honest majority [10, 18, 64, 7, 6, 26, 11, 8, 25].

While the special case of MPC with dishonest majority, namely the two-party computation (2PC) has been at the focus of numerous works [67, 46, 54, 66, 42, 1, 65, 59], 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 [15] was designed for the three-party setting. Several other applications solved via 3PC include statistical data analysis [14], email-filtering [52], financial data analysis [14] and distributed credential encryption service [60]. The practical efficiency of 3PC has thus got considerable emphasis in the past and some of them have evolved to technologies [33, 13, 53, 52, 20, 30, 3]. 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 [22]. 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 [35] for *fair* MPC with $t \geq 2$ and arbitrary number of parties, even in the presence of broadcast channels. [43] 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 [34, 62, 5]. Interestingly enough, a perfect VSS with $(n = 5, t = 1)$ [34], statistical VSS with $(n = 4, t = 1)$ [62, 43] and cryptographic VSS with $(n = 4, t = 1)$ [5] 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 [43, 60, 17], 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 [43, 60] allows to circumvent certain inherent challenges in malicious 2PC such as enforcing correctness of garbling which incurs additional communication.

The exact round complexity is yet another measure that sets apart the protocols with three parties over the ones with two parties. For instance, 3PC protocol is achievable just in two rounds with the minimal network setting of pairwise-private channels [43]. The

2PC (and MPC with dishonest majority) protocols achieving the same level of security (with abort) necessarily require 4 rounds [50] and have to resort to a common reference string (CRS) to shoot for the best possible round complexity of 2 [41].

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) [41]. In the plain model, any actively secure 2PC is known to require 5 rounds in non-simultaneous message model [50] (under black-box simulation). The bound can be improved to 4 even in the dishonest majority setting [32] in simultaneous message model and tight upper bounds are presented in [16, 2, 40]. With a common reference string (CRS), the lower bound can be further improved to 2 rounds [41]. Tight upper bounds are shown in [31] under indistinguishability obfuscation (assumption weakened to witness encryption by [39]), and in [61] under a variant of Fully Homomorphic Encryption (FHE) and Non-interactive Zero-knowledge.

In the honest majority setting which is shown to be necessary [22] and sufficient [10, 18, 24] 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 [39]. 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 [35]. In an interesting work, [43] 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), [43] presents a 2-round 3PC protocol tolerating single corruption; whose communication and computation efficiency was improved by the 3-round protocol of [60]. Both these protocols achieve a weaker notion of security known as security with selective abort. Selective abort security [44] (referred as 'security with abort and no fairness' in [38]) 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 [38]), either all or none of the honest parties receive the output. An easy observation concludes that the 3PC of [60] 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 [30, 3]. [21] studies 3PC with dishonest majority. Recently, [17] presents a practically efficient 5-party MPC protocol in honest majority setting, going beyond 3-party case, relying on distributed garbling technique based on [7].

Ref.	Setting	Round	Network Setting / Assumption	Security	Comments
[4]	$t < n/2$	≥ 5	private channel, Broadcast / CRS, FHE, NIZK	fairness	upper bound
[39]	$t < n/2$	3	non-private channel, Broadcast / CRS, FHE	guaranteed output delivery	upper bound
[39]	$t < n/2$	2	non-private channel, Broadcast / CRS, PKI, FHE	guaranteed output delivery	upper bound
[44]	$n = 5, t = 1$	2	private channel / OWF	guaranteed output delivery	upper bound
[43]	$n = 3, t = 1$	2	private channel / OWF	selective abort	upper bound
[43]	$n = 4, t = 1$	2	private channel / (injective) OWF	guaranteed output delivery	upper bound
[60]	$n = 3, t = 1$	3	private channel, Broadcast / PRG	unanimous abort	upper bound
[39]	$t < n/2$	3	non-private channel, Broadcast / CRS	fairness	lower bound
[35]	$n; t > 1$	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 [43], while guaranteed output delivery is infeasible to achieve irrespective of the number of rounds [23]. No bound on round complexity is known for unanimous abort or fairness. In the setting with a broadcast channel, the result of [60] implies 3-round 3PC with unanimous abort. Neither the round optimality of the [60] 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 [43] and complement the lower bound of [35] 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 [21, 43, 60]. 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 [43] 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) [36], 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 [60]. 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 [35] 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 [43] an optimal one. Notably, our result does not contradict with the two-round protocol of [39] 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 [41] 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	[41] / [43]	2	[41] / [43]
Unanimous Abort	3	**This paper / This paper**	2	[41] / **This paper**
Fairness	3	**This paper / This paper**	3	**This paper / This paper**
Guaranteed output delivery	Impossible	[23]	3	**This paper / This paper**

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 pairwise channels. The lower bounds are shown by taking a special 3-party function and by devising a sequence hybrid executions under different adversarial strategies, allowing to conclude any 3PC protocol computing the considered function cannot be simultaneously private and fair or secure with unanimous abort.

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 [45] or identifiable fairness to guaranteed output delivery [24], 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 [60], 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 [43]). 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 [54, 21]. The issue of input consistency does not appear in the construction of [60] at all, as it does not deal with parallel composition. On the other hand, the generic input consistency technique adopted in [43] 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 [43, 60]. 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 [36] 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 [49, 29]. 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, [47] 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 [54] 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' [58, 51] and breach privacy of his honest co-garbler. We tackle this using 'XOR-tree approach' [55] 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, [55, 66] 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 [67] 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 [49, 29]). 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. 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. Our lower bound results appear in Section 6. The security definitions, complete security proofs and optimizations appear in the full version [63]. We define authenticated CDS and show its realisation from one of the sub-protocols used in our fair protocol in the full version.

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 κ. A negligible function in κ is denoted by $\texttt{negl}(\kappa)$. A function $\texttt{negl}(\cdot)$ is *negligible* if for every polynomial $p(\cdot)$ there exists a value N such that for all $m > N$ it holds that $\texttt{negl}(m) < \frac{1}{p(m)}$. We denote by $[x]$, the set of elements $\{1, \ldots, x\}$ and by $[x, y]$ for $y > x$, the set of elements $\{x, x+1, \ldots, y\}$. 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 [7], but it had largely only been a technique used in secure protocols until they were formalized as a primitive by Bellare et al. [9]. 'Garbling Schemes' as they were termed, were assigned well-defined notions of security, namely *correctness*, *privacy*, *obliviousness*, and *authenticity*. A garbling scheme $\mathcal{G}$ is characterised by a tuple of PPT algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ described below.

- $\mathsf{Gb}\,(1^\kappa, C)$ is invoked on a circuit C in order to produce a 'garbled circuit' $\mathbf{C}$, 'input encoding information' e, and 'output decoding information' d.
- $\mathsf{En}\,(x, e)$ encodes a clear input x with encoding information e in order to produce a garbled/encoded input $\mathbf{X}$.
- $\mathsf{Ev}\,(\mathbf{C}, \mathbf{X})$ evaluates $\mathbf{C}$ on $\mathbf{X}$ to produce a garbled/encoded output $\mathbf{Y}$.
- $\mathsf{De}\,(\mathbf{Y}, d)$ 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 presented in the full version [63].

We are interested in a class of garbling schemes referred to as *projective* in [9]. When garbling a circuit $C : \{0,1\}^n \mapsto \{0,1\}^m$, a projective garbling scheme produces encoding information of the form $e = \left(e_i^0, e_i^1\right)_{i\in[n]}$, and the encoded input $\mathbf{X}$ for $x = (x_i)_{i\in[n]}$ can be interpreted as $\mathbf{X} = \mathsf{En}(x, e) = (e_i^{x_i})_{i\in[n]}$.

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* [49, 29]. 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 sDe [60] that can decode garbled outputs without the decoding information d. The soft-decoding algorithm must comply with correctness: $\mathsf{sDe}(\mathsf{Ev}(\mathbf{C}, \mathsf{En}(e, x))) = C(x)$ for all $(\mathbf{C}, e, d)$. While both sDe and De can decode garbled outputs, the authenticity needs to hold only with respect to De. In practice, soft decoding in typical garbling schemes can be achieved by simply appending the truth value to each output wire label.

Non-interactive Commitment Schemes. A non-interactive commitment scheme (NICOM) consists of two algorithms $(\mathsf{Com}, \mathsf{Open})$ defined as follows. Given a security parameter κ, a common parameter pp, message x and random coins r, PPT algorithm Com outputs commitment c and corresponding opening information o. Given κ, pp, a commitment and corresponding opening information (c, o), PPT algorithm Open outputs the message x. The algorithms should satisfy correctness, binding (i.e. it must be hard for an adversary to come up with two different openings of any c and *any* pp) and hiding (a commitment must not leak information about the underlying message) properties. We need this kind of strong binding as the same party who generates the pp and commitment is required to open later. Two such instantiations of NICOM based on symmetric key primitives (specifically, injective one-way functions) and the formal definitions of the properties are given in the full version. We also need a NICOM scheme that admits equivocation property. An equivocal non-interactive commitment (eNICOM) is a NICOM that allows equivocation of a certain commitment to any given message with the help of a trapdoor. The formal definitions and instantiations appear in the full version [63].

Symmetric-Key Encryption (SKE) with Special Correctness. Our fair protocol uses a SKE $\pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which satisfies CPA security and a special correctness property [48, 56]– if the encryption and decryption keys are different, then decryption fails with high probability. The definition and an instantiation appear in the full version.

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 lower bound result showing necessity of three rounds for unanimous abort assuming just pairwise private channels (appears in the full version [63]) 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) fair_i where P_i acts as the evaluator and the other two as garblers for computing the desired function f. This sub-protocol ensures that honest P_i either computes its encoded output or identifies just a conflict in the worst case. The decoding information is committed to P_i, yet not opened. It is released in Round 3 of the final composed protocol under subtle conditions as elaborated below. (b) cert_i where P_i acts as the evaluator and the other two as garblers for computing an equality checking circuit on the common information distributed by P_i in the first round of the final protocol. Notably, though the inputs come solely from the garblers, they are originated from the evaluator and so the circuit can be garbled in a privacy-free fashion. This sub-protocol ensures either honest P_i gets its certificate, the key for output 1 (meaning the equality check passes through), or identifies a conflict in the worst case. The second round of cert_i is essentially an 'authenticated' CDS for equality predicate tolerating one active corruption. Three *global* variables are maintained by each party P_i to keep tab on the conflicts and the corrupt. Namely, $\mathcal{C}_i$ to keep the identity of the corrupt, flag_j and flag_k (for distinct $i, j, k \in [3]$) as indicators of detection of conflict with respect to information distributed by P_j and P_k respectively. The sub-protocols fair_i and cert_i assure that if neither the two flags nor $\mathcal{C}_i$ is set, then P_i must be able to evaluate the GC successfully and get its certificate respectively.

Once $\{\mathsf{fair}_i, \mathsf{cert}_i\}_{i\in[3]}$ complete by the end of round 2 of the final protocol fair, any honest party will be in one of the three states: (a) no corruption and no conflict detected ($(\mathcal{C}_i = \emptyset) \wedge (\mathsf{flag}_j = 0) \wedge (\mathsf{flag}_k = 0)$); (b) corruption detected ($\mathcal{C}_i \neq \emptyset$); (c) conflict detected $(\mathsf{flag}_j = 1) \vee (\mathsf{flag}_k = 1)$. An honest party, guaranteed to have computed its encoded output and certificate *only* in the first state, releases these as well as the decoding information for both the other parties unconditionally in the third round. In the other two states, an honest party conditionally releases only the decoding information. This step is extremely crucial for maintaining fairness. Specifically, a party that belongs to the second state, releases the decoding information only to the party identified to be honest. A party that belongs to the third state, releases the decoding information in encrypted form *only* to the party whose distributed information are not agreed upon, so that the encryption can be unlocked only via a valid certificate. A corrupt party will either have its certificate or the encrypted decoding information, but *not* both. The former when it distributes its common information correctly and the latter when it does not. The only way a corrupt party can get its decoding information is by keeping one honest party in the first state, in which case both the honest parties will be able to compute the output as follows. The honest party in state one, say P_i, either gets it decoding information on clear or in encrypted form. The former when the other honest party, P_j is in the first or second state and the latter when P_j is in the third state. P_i retrieves the decoding information no matter what, as it also holds the certificate to open the encryption. An honest party P_j in the second state, on identifying P_i as honest, takes the encoded output of P_i and uses its own decoding information to compute the output. The case for an honest party P_j in the third state is the most interesting. Since

honest P_i belongs to the first state, a corrupt party must have distributed its common information correctly as otherwise P_i will find a conflict and would be in third state. Therefore, P_j in the third state must have found P_i's information on disagreement due the corrupt party's misbehaviour. Now, P_i's certificate that proves his correct behaviour, allows P_j to identify the corrupt, enter into the second state and compute the output by taking the encoded output of honest P_i. In the following, we describe execution fair_i assuming input consistency, followed by cert_i. Entwining the six executions, tackling the input consistency and the final presentation of protocol fair appear in the end.

3.1 Protocol fair_i

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

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

Protocol fair_i appears in Figure 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 $C_j^{\mathsf{dec}}, C_k^{\mathsf{dec}}$ and the flags $\mathsf{flag}_j, \mathsf{flag}_k$ (initially set to false) to be used in the final protocol. The garblers output their respective corrupt set, flag for the fellow garbler and opening for the decoding information corresponding to its

co-garbler's GC and *not* its own. This is to ensure that it cannot break the binding of eNICOM which may not necessarily hold for adversarially-picked public parameter.

Lemma 1. *During* fair_i, $P_\beta \notin \mathcal{C}_\alpha$ *holds for honest* P_α, P_β.

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

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

Proof. According to 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 flag_j or flag_k to true. For the third condition, if P_j sends `nOK` then P_i would set $\mathsf{flag}_k = 1$. Lastly, if P_k sends `nOK`, then P_i sets $\mathsf{flag}_j = 1$. Clearly when $\mathcal{C}_i = \emptyset \wedge \mathsf{flag}_j = 0 \wedge \mathsf{flag}_k = 0$, P_i evaluates both $\mathbf{C}_j, \mathbf{C}_k$ and obtains $\mathbf{Y}_i = (\mathbf{Y}_i^j, \mathbf{Y}_i^k) \neq \perp$. □

3.2 Protocol cert_i

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

Similar to fair_i, at the end of the protocol, either P_i gets its certificate (either the key for 1 or the inputs themselves), or sets its flags (when GC and commitment do not match) or sets its corrupt set (when opening of encoded inputs fail). P_i outputs its certificate, the flag for the GC generator and corrupt set, to be used in the final protocol.

Protocol $\mathsf{fair}_i()$

Inputs: Party P_α has x_α for $\alpha \in [3]$.
Common Inputs: The circuit $C(x_1, x_2, x_3, x_4)$ that computes $f(x_1, x_2, x_3 \oplus x_4)$.
Output: A garbler P_l ($l \in \{j,k\}$) outputs corrupt set $\mathcal{C}_l, \mathsf{flag}_{\{j,k\}\setminus l}$ and O_i^{dec}. P_i outputs ($\mathcal{C}_i$, $\mathbf{Y}_i = (\mathbf{Y}_i^j, \mathbf{Y}_i^k), C_j^{\mathsf{dec}}, C_k^{\mathsf{dec}}, \mathsf{flag}_j, \mathsf{flag}_k$) where $\mathbf{Y}_i$ denote a pair of encoded outputs or $\perp$.
Primitives: A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private and oblivious, a NICOM $(\mathsf{Com}, \mathsf{Open})$, an eNICOM $(\mathsf{eGen}, \mathsf{eCom}, \mathsf{eOpen}, \mathsf{Equiv})$ and a PRG G.

Round 1:

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

Round 2:

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

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

Fig. 1: Protocol fair_i

The garblers output the key for 1, flag for its fellow garbler and the corrupt set. Notice that, when cert_i is composed in the bigger protocol, P_i will be in a position to identify the corrupt when the equality fails and the certificate is the inputs fed by the garblers. The protocol appears in Figure 2.

$\mathsf{cert}_i()$

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

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

Output: A garbler P_l for $l \in \{j, k\}$ outputs corrupt set $\mathcal{C}_l$ and $\mathbf{key}_i$. P_i outputs $(\mathbf{cert}_i, \mathcal{C}_i, \mathsf{flag}_j, \mathsf{flag}_k)$. Garbler P_k additionally outputs flag_j.

Round 1: P_j does the following:

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

Round 2:

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

Fig. 2: Protocol cert_i

Lemma 3. *During* cert_i, $P_\beta \notin \mathcal{C}_\alpha$ *holds for honest* P_α, P_β.

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

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

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

□

3.3 Protocol fair

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

We tackle the former in a simple yet clever way without incurring any additional overhead. We explain the technique for enforcing P_1's input consistency on input x_1 as an evaluator during fair_1 and as a garbler during $\mathsf{fair}_2, \mathsf{fair}_3$ with respect to his GC $\mathbf{C}_1$. 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 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 fair_2, P_1 and P_3 who act as the garblers use x_{13} as the permutation vector p_{11} that defines the order of the commitments of the bits of x_1. Now input consistency of P_1's input is guaranteed if m_{11} transferred by P_1 in fair_2 is same as x_{12}, P_1's share for P_2 in fair_1. For an honest P_1, the above will be true since $m_{11} = p_{11} \oplus x_1 = x_{13} \oplus x_1 = x_{12}$. If the check fails, then P_2 identifies P_1 as corrupt. This simple check forces P_1 to use the same input in both fair_1 and fair_2 (corresponding to $\mathbf{C}_1$). A similar trick is used to ensure input consistency of the input of P_1 across fair_1 and fair_3 (corresponding to $\mathbf{C}_1$) 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 fair_3 checks if m_{11} transferred by P_1 in fair_3 is same as x_{13} that P_3 receives from P_1 in fair_1. While the above technique enforces the consistency with respect to P_1's GC, unfortunately, the same technique cannot be used to enforce P_1's input consistency with respect to $\mathbf{C}_2$ in fair_3 (or fair_2) since p_{21} cannot be set to x_{12} which is available to P_2 only at the end of first round. While, P_2 needs to prepare and broadcast the commitments to the encoding information in jumbled order as per permutation string p_{21} in the first round itself. We handle it differently as below.

The consistency of P_i's input for both garbled circuits in fair_j (and similarly in fair_k) is tackled via 'cheat-recovery mechanism' [54]. We explain with respect to P_1's input in fair_3. P_2 prepares a ciphertext (cheat recovery box) with the input keys of P_1 corresponding to the mismatched input bit in the two garbled circuits, $\mathbf{C}_1$ and $\mathbf{C}_2$ in fair_3. This ciphertext encrypts the the input shares of garblers that P_3 misses, namely, x_{12} and x_{21}. This would allow P_3 to compute the function on clear inputs directly. To ensure that the recovered missing shares are as distributed in fair_1 and fair_2, the shares are not simply distributed but are committed via NICOM by the input owners and the openings are encrypted by the holders. Since there is no way for an evaluator to detect any mismatch in the inputs to and outputs from the two GCs as they are in encoded form, we use encryption scheme with special correctness 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 $\mathbf{C}_1$ and $\mathbf{C}_2$ can conclude whether its two different inputs lead to the same output or not, breaching privacy. Note that the decoding information cannot be

given via this cheat recovery box that uses conflicting encoded outputs as key, as that would result in circularity.

Despite using the above fix, the mechanism as discussed above is susceptible to 'selective failure attack', an attack well-known in the 2-party domain. While in the latter domain, the attack is launched to breach the privacy of the evaluator's input based on whether it aborts or not. Here, a corrupt garbler can prepare the ciphertexts in an incorrect way and can breach privacy of its honest co-garbler based on whether clear or encoded output comes out of the evaluator. We elaborate the attack in fair_3 considering a corrupt P_1 and single bit inputs. P_1 is supposed to prepare two ciphertexts corresponding to P_2's input bit using the following key combinations– (a) key for 0 in $\mathbf{C}_1$ and 1 in $\mathbf{C}_2$ and (b) vice-versa. Corrupt P_1 may replace one of the ciphertexts using key based on encoded input 0 of P_2 in both the GCs. In case P_2 indeed has input 0 (that he would use consistently across the 2 GCs during fair_3), then P_3 would be able to decrypt the ciphertext and would send clear output in Round 3. P_1 can readily conclude that P_2's input is 0. This attack is taken care via the usual technique of breaking each input bit to s number of xor-shares, referred as 'XOR-tree approach' [55] (probe-resistance matrix [55, 66] 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.

We now prove the correctness of fair. The intuitive proof of fairness and formal proof of security are presented in the full version [63].

Lemma 5. *During* fair*,* $P_j \notin \mathcal{C}_i$ *holds for honest* P_i, P_j*.*

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

In Round 1 of fair, P_i includes P_j as corrupt only if (a) P_i, P_j are garblers and P_j sets $p_{jj} \neq x_{ji}$ or (b) P_j sends $\mathsf{pp}_j, c_{ji}, o_{ji}, x_{ji}$ to P_i such that $\mathsf{Open}(\mathsf{pp}_j, c_{ji}, o_{ji}) \neq x_{ji}$. 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 $\mathbf{cert}_i = (\gamma'_j, \gamma'_k)$ and detects P_j's input γ'_j in cert_i to be different from the information sent by him. The former will not be true for an honest P_j. The latter also cannot hold for honest P_j by correctness of the privacy-free garbling used. In the last round of fair, P_i will identify P_j as corrupt, if it has $\mathsf{flag}_k = 1$ and yet receives $\mathbf{cert}_k$ which is same as $\mathbf{key}_k$ from P_k. A corrupt P_k receives $\mathbf{key}_k$ 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

Protocol $\mathsf{fair}()$

Inputs: Party P_i has x_i for $i \in [3]$.
Output: $y = f(x_1, x_2, x_3)$ or $\perp$ where the inputs and the function output belong to $\{0,1\}^\ell$.
Subprotocols: fair_i for $i \in [3]$ (Figure 1), cert_i for $i \in [3]$ (Figure 2), SKE $(\mathsf{Enc}, \mathsf{Dec})$ with 'special correctness'.

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

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

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

- If $P_i \notin \mathcal{C}_j$, P_j sends $(\mathsf{pp}_i, c_{ij}, c_{ik})$ to P_k. If $P_i \notin \mathcal{C}_k$, P_k sends $(\mathsf{pp}_i, c_{ij}, c_{ik})$ to P_j. They set $\mathsf{flag}_i = 1$ in case of mismatch or no communication.
- If $P_i \notin \mathcal{C}_j$, P_j participates in cert_i as a garbler with input γ_j as $\{\mathcal{D}_i^j, \mathcal{D}_i^k, \mathcal{W}_k, \mathsf{pp}_i, c_{ij}, c_{ik}\}$ where $\mathcal{D}_i^j, \mathcal{D}_i^k, \mathcal{W}_k$ and $(\mathsf{pp}_i, c_{ij}, c_{ik})$ was received from P_i during Round 1 of $\mathsf{fair}_j, \mathsf{fair}_k$, cert_k (assuming $k = (i+1) \mod 3$) and fair respectively. Similar step is taken by P_k.
- If $\mathbf{cert}_i = (\gamma'_j, \gamma'_k)$, P_i sets $\mathcal{C}_i = P_l$ if $\gamma'_l \neq \{\mathcal{D}_i^j, \mathcal{D}_i^k, \mathcal{W}_k, \mathsf{pp}_i, c_{ij}, c_{ik}\}$ for $l \in \{j, k\}$.
- If $P_i \notin \mathcal{C}_j$, P_j participates in **Round 2** of fair_i. When $P_k \notin \mathcal{C}_j$, P_j additionally sends the ciphertexts $\mathsf{ct}_{j\alpha}^\beta$ for $\beta \in \{0,1\}$ and $\alpha \in [\ell]$ created as follows. Let $\{\mathbf{X}_{l(\ell+\alpha)}^0, \mathbf{X}_{l(\ell+\alpha)}^1\}$, denote the encoding information of co-garbler P_k's input wire α corresponding to $\mathbf{C}_l$ $(l \in \{j,k\})$. Then $\mathsf{ct}_{j\alpha}^\beta = \mathsf{Enc}_{\mathsf{sk}_\alpha^\beta}(o_{jk}, o_{kj})$ for $\mathsf{sk}_\alpha^0 = \mathbf{X}_{j(\ell+\alpha)}^0 \oplus \mathbf{X}_{k(\ell+\alpha)}^1$ and $\mathsf{sk}_\alpha^1 = \mathbf{X}_{j(\ell+\alpha)}^1 \oplus \mathbf{X}_{k(\ell+\alpha)}^0$. P_k takes similar steps.
- (Local Computation by P_i) Include P_l in $\mathcal{C}_i$ if $m_{ll} \neq x_{li}$ for $l \in \{j,k\}$. If $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$, then use key $\mathbf{X}_{j(\ell+\alpha)}^{m_{jk}^\alpha} \oplus \mathbf{X}_{k(\ell+\alpha)}^{m_{kk}^\alpha}$ $(\alpha \in [\ell])$ to decrypt the ciphertexts $\mathsf{ct}_{j\alpha}^0$ or $\mathsf{ct}_{j\alpha}^1$ obtained from P_j. If the decryption succeeds, retrieve o_{kj}, o_{jk}. Execute $x_{kj} \leftarrow \mathsf{Open}(c_{kj}, o_{kj})$ and $x_{jk} \leftarrow \mathsf{Open}(c_{jk}, o_{jk})$. If the opening succeeds, then evaluate f on $(x_i, x_{ji} \oplus x_{jk}, x_{ki} \oplus x_{kj})$ to obtain y. Similarly, steps are taken with respect to P_j's input, using the key $\mathbf{X}_{j\alpha}^{m_{jj}^\alpha} \oplus \mathbf{X}_{k\alpha}^{m_{kj}^\alpha}$ to decrypt the ciphertexts $\mathsf{ct}_{k\alpha}^0$ or $\mathsf{ct}_{k\alpha}^1$ obtained from P_k.

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

be true for P_k to obtain $\mathbf{key}_k$ (except for the case when it breaks the authenticity of the GC): (i) γ_i and γ_j for cert_k must be same and (ii) P_k must not be in the corrupt set of any honest party at the end of Round 1. In this case, flag_k cannot be 1. □

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

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

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

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

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

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

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

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

Lemma 7. *If an honest party is in* $\{\mathtt{st}_1, \mathtt{st}_2\}$*, then corrupt party must have committed a unique input.*

Proof. An honest P_i is in $\{\mathtt{st}_1, \mathtt{st}_2\}$ only when $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$ hold at the end of Round 2. Assume P_k is corrupt. P_k has not committed to a unique x_k implies either it has distributed different copies of commitments (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 $\mathcal{C}_i \neq \emptyset$. Otherwise, P_j populates its corrupt set with P_k, leading to P_i setting $\mathsf{flag}_k = 1$ in Round 2. □

Lemma 8. *If an honest party is in* $\mathtt{st}_1$*, then its output* y *corresponds to the unique input committed by the corrupt party.*

Proof. An honest P_i is in $\mathtt{st}_1$ only when $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$ hold at the end of Round 2 and it computes y via decryption of the ciphertexts ct sent by either P_j or P_k. Assume P_k is corrupt. By Lemma 7, P_k has committed to its input. The condition $\mathsf{flag}_j = 0$ implies that P_k exchanges the commitments on the shares of P_j's input, namely $\{c_{ji}, c_{jk}\}$, 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 $x_{kj} \leftarrow \mathsf{Open}(c_{kj}, o_{kj})$ and $x_{jk} \leftarrow \mathsf{Open}(c_{jk}, o_{jk})$ 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. □

Lemma 9. *If an honest party is in* $\mathtt{st}_2$*, then its encoded output* $\mathbf{Y}$ *corresponds to the unique input committed by the corrupt party.*

Proof. An honest P_i is in $\mathtt{st}_2$ only when $\mathcal{C}_i = \emptyset$, $\mathsf{flag}_j = 0, \mathsf{flag}_k = 0$ hold at the end of Round 2. The conditions also imply that P_i has computed $\mathbf{Y}_i$ 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 fair_i. Otherwise one of the ciphertexts prepared by honest P_j must have been opened and y would be computed, implying P_i belongs to $\mathtt{st}_1$ and not in $\mathtt{st}_2$ as assumed. We are now left to show that the input of P_k for its circuit $\mathbf{C}_k$ in fair_i is the same as the one committed.

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

□

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

Proof. Note that an honest party P_i in $\mathtt{st}_2$ either uses y of another party in $\mathtt{st}_1$ or computes output from its encoded output $\mathbf{Y}_i$. The proof for the former case goes as follows. By Lemma 6, a corrupt P_k can never be in $\mathtt{st}_1$. The correctness of y computed by an honest P_j follows directly from Lemma 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 $\mathsf{flag}_j = \mathsf{flag}_k = 0$ 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). □

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

Proof. An honest party P_i in $\mathtt{st}_3$ either uses y of another party in $\mathtt{st}_1$ or computes output from encoded output $\mathbf{Y}_j$ 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 $\mathtt{st}_4$ only uses y of another party in $\mathtt{st}_1$. The lemma follows in this case via the same argument as before. □

Theorem 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 $\{\mathtt{st}_1, \mathtt{st}_2, \mathtt{st}_3, \mathtt{st}_4\}$ at the time of output computation. The proof now follows from Lemmas 7,8,10,11. □

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 [41]. Our lower bound result presented in the full version [63] 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 [45] 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 [43] 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 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. The formal description and proof of security of ua appear in the full version [63].

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 [23].

Our tryst starts with the known generic transformations that are relevant such as the transformations from the unanimous abort to (identifiable) fair protocol [45] or identifiable fair to guaranteed output delivery [24]. 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. The protocol and the proof appear in the full version [63].

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 [57]. 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)**). The first lower bound appears below. We prove the second lower bound in the full version [63].

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. [39] 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). [35] 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 [39] and [35] (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 [62] (since VSS is a special case of MPC). We now prove the impossibility formally.

Theorem 2. *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 π that can compute $f(x_1, x_2, x_3)$ defined below for P_i's input x_i:

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

At a high level, we discuss two adversarial strategies $\mathcal{A}_1$ and $\mathcal{A}_2$ of $\mathcal{A}$. We consider party P_i launching $\mathcal{A}_i$ in execution Σ_i ($i \in [2]$) of π. Both the executions are assumed to be run for the same input tuple (x_1, x_2, x_3) and the same random inputs (r_1, r_2, r_3)

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

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

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

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

Next we present the views of the parties in the two executions Σ_1 and Σ_2 in Table 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. $\mathsf{p}^2_{1\to 3}(\star)$). We now prove a sequence of lemmas to complete our proof.

Lemma 12. *A corrupt P_1 launching $\mathcal{A}_1$ in Σ_1 should learn the output $y = f(x_1, x_2, x_3)$.*

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

Table 1: Views of P_1, P_2, P_3 in Σ_1 and Σ_2

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

Lemma 13. *A corrupt P_2 launching $\mathcal{A}_2$ in Σ_2 should learn the output y.*

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

Lemma 14. *P_3 in Σ_2 should learn the output y by the end of Round 1.*

Proof. According to the previous lemma, P_2 should learn the function output in Σ_2. Due to 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 $\{\mathsf{p}^1_{1\to3}, \mathsf{b}^1_1\}$ of P_1 does not depend on x_2. Next, since there is no private communication to P_1 from P_2 as per strategy $\mathcal{A}_2$, the only information that can possibly hold information on x_2 and can impact the round 2 messages of P_1 is b^1_2. However, since this is a broadcast message, P_3 holds this by the end of Round 1 itself. □

Lemma 15. *A corrupt P_3 violates the privacy property of π.*

Proof. The adversary corrupting P_3 participates in the protocol honestly by fixing input $x_3 = 0$. Since P_3 can get the output from P_2's and P_1's round 1 communication (Lemma 14), it must be true that P_3 can evaluate the function f locally by plugging in any value of x_3. (Note that P_2 and P_1's communication in round 1 are independent of the communication of P_3 in the same round.) Now a corrupt P_3 can plug in $x_3 = 1$ locally and learn x_2 (via the output $x_2 \wedge x_3$). In the ideal world, corrupt P_3 must learn nothing beyond the output 0 as it has participated in the protocol with input 0. But in the execution of π (in which P_3 participated honestly with input $x_3 = 0$), P_3 has learnt x_2. This is a clear breach of privacy as P_3 learns x_2 regardless of his input.

□

Hence, we have arrived at a contradiction, completing the proof of Theorem 2.

□

Acknowledgement. The first author would like to acknowledge partial support from SERB Women Excellence Award from Science and Engineering Research Board of India.

References

1. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT*, 2014.
2. Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In *CRYPTO*, 2017.
3. 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.
4. 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.
5. Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *ASIACRYPT*, 2011.
6. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.
7. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *ACM STOC*, 1990.
8. Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In *TCC*, 2006.
9. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, 2012.
10. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, 1988.
11. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *CRYPTO*, 2012.
12. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, 2011.
13. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security- ESORICS*, 2008.
14. Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis - (short paper). In *FC*, 2012.
15. 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.
16. Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In *TCC*, 2017.
17. 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.
18. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *ACM STOC*, 1988.
19. David Chaum, Ivan Damgård, and Jeroen Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO*, 1987.

20. 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.
21. Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *CRYPTO*, 2014.
22. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *ACM STOC*, 1986.
23. Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. Characterization of secure multiparty computation without broadcast. In *TCC*, 2016.
24. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT*, 2014.
25. Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT*, 1999.
26. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO*, 2007.
27. Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, 2010.
28. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.
29. Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT*, 2015.
30. 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.
31. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, 2014.
32. Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In *EUROCRYPT*, 2016.
33. Martin Geisler. Viff: Virtual ideal functionality framework, 2007.
34. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *ACM STOC*, 2001.
35. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *CRYPTO*, 2002.
36. Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. *J. Comput. Syst. Sci.*, 2000.
37. 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.
38. Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *DISC*, 2002.
39. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, 2015.
40. Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. Cryptology ePrint Archive, Report 2017/1056, 2017. https://eprint.iacr.org/2017/1056.
41. Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, 2011.
42. Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J. Malozemoff. Amortizing garbled circuits. In *CRYPTO*, 2014.
43. Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO*, 2015.

44. Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, 2010.
45. Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *CRYPTO*, 2016.
46. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, 2008.
47. Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *ICALP*, 2014.
48. Zahra Jafargholi and Daniel Wichs. Adaptive security of yao's garbled circuits. In *TCC 2016-B*, 2016.
49. Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *CCS*, 2013.
50. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *CRYPTO*, 2004.
51. 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.
52. John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-scale secure multiparty computation. In *ESOP*, 2014.
53. 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.
54. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, 2013.
55. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.
56. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 2009.
57. N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
58. Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, 2006.
59. Payman Mohassel and Mike Rosulek. Non-interactive secure 2pc in the offline/online and batch settings. In *EUROCRYPT*, 2017.
60. Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *ACM CCS*, 2015.
61. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, 2016.
62. Arpita Patra, Ashish Choudhary, Tal Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In *CRYPTO*, 2009.
63. Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. Cryptology ePrint Archive, Report 2018/481, 2018. https://eprint.iacr.org/2018/481.
64. Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *ACM STOC*, 1989.
65. Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *USENIX Security Symposium*, 2016.
66. Abhi Shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In *ACM CCS*, 2013.
67. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982.