# Fast Secure Computation for Small Population over the Internet

Megha Byali
Indian Institute of Science
Bangalore, India
megha@iisc.ac.in

Arun Joseph
Indian Institute of Science
Bangalore, India
arunj@iisc.ac.in

Arpita Patra*
Indian Institute of Science
Bangalore, India
arpita@iisc.ac.in

Divya Ravi
Indian Institute of Science
Bangalore, India
divyar@iisc.ac.in

## ABSTRACT

Secure Multi-Party Computation (MPC) with small number of parties is an interesting area of research, primarily due to its ability to model most real-life MPC applications and the simplicity and efficiency of the resulting protocols. In this work, we present efficient, *constant-round* 3-party (3PC) and 4-party (4PC) protocols in the honest-majority setting that achieve strong security notions of *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). Being constant-round, our constructions are suitable for Internet-like high-latency networks and are built from garbled circuits (GC).

Assuming the minimal model of pairwise-private channels, we present two protocols that involve computation and communication of a *single* GC– (a) a 4-round 3PC with fairness, (b) a 5-round 4PC with guaranteed output delivery. Empirically, our protocols are on par with the best known 3PC protocol of Mohassel et al. [CCS 2015] that only achieves security with selective abort, in terms of the computation time, LAN runtime, WAN runtime and communication cost. In fact, our 4PC outperforms the 3PC of Mohassel et al. significantly in terms of per-party computation and communication cost. With an extra GC, we improve the round complexity of our 4PC to four rounds. The only 4PC in our setting, given by Ishai et al. [CRYPTO 2015], involves 12 GCs.

Assuming an additional broadcast channel, we present a 5-round 3PC with guaranteed output delivery that involves computation and communication of a *single* GC. A broadcast channel is inevitable in this setting for achieving guaranteed output delivery, owing to an impossibility result in the literature. The overall broadcast communication of our protocol is nominal and most importantly, is independent of the circuit size. This protocol too induces a nominal overhead compared to the protocol of Mohassel et al.

*This author would like to acknowledge financial support by SERB Women Excellence Award from Science and Engineering Research Board of India.


*CCS '18, October 15–19, 2018, Toronto, ON, Canada*

ACM ISBN 978-1-4503-5693-0/18/10…$15.00
https://doi.org/10.1145/3243734.3243784

## CCS CONCEPTS

• **Theory of computation → Cryptographic protocols**;

## KEYWORDS

Secure Multiparty Computation; Fairness; Guaranteed Output Delivery; Garbled Circuits

**ACM Reference Format:**
Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. 2018. Fast Secure Computation for Small Population over the Internet. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243784

## 1 INTRODUCTION

Secure Multi-Party Computation (MPC) [22, 36, 57], the standard bearer problem of cryptography, has evolved tremendously over recent years. It studies the problem of enabling a set of $n$ parties to perform joint computation on their private inputs, in a way that no coalition of $t$ parties can learn more information than the output (privacy) or affect the true output of the computation (correctness). While the vast literature in MPC has traditionally been focused on theoretical aspects, lately, with increasing demand for efficient constructions suitable in real-time applications, there has been a growing interest to improve the concrete efficiency of protocols.

The domain of MPC can be broadly classified into honest majority [6–9, 13, 14, 29, 55] and dishonest majority [5, 15, 30, 31, 33, 36, 49] settings. The special case of two-party (2PC) in dishonest majority setting has enjoyed overwhelming focus over the years in terms of improving its efficiency [1, 45, 47, 48, 53]. In contrast, the special cases of honest majority setting have not been in the limelight until recently when practically efficient MPC constructions of [4, 21, 32, 51] leveraged presence of small number of parties.

The area of MPC with small number of parties in the honest majority setting has drawn popularity particularly due to simplicity and efficiency of the resulting protocols. The area is worthwhile to explore due to the following reasons. First, most real-time applications involve small number of parties. For instance, applications such as statistical data analysis [17], email-filtering [44], financial data analysis [17], distributed credential encryption [51], Danish sugar beet auction [18] involve 3 parties. Additionally, the MPC frameworks such as VIFF [34], Sharemind [16] have been explored with 3 parties. The recent advances that involve MPC in privacy-preserving machine learning have exhibited applications

that involve small number of parties [50]. Second, for the instance of large-scale computation on data involving sensitive information, it is preferable to utilize 3 or 4 servers in comparison to only two for improved fault-tolerance and performance. Another crucial advantage over the 2-party setting is enabling stronger 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 attainable only in the honest majority setting [24]. These goals are desirable in practical applications as they serve as a stimulant for parties to be engaged in the computation. Also, having honest majority is advantageous to obtain constructions relying on weaker cryptographic assumptions and light-weight cryptographic tools. For example, the protocols of [40, 51] are built using symmetric-key primitives whereas 2PC protocols require Oblivious Transfer (OT) [41, 47, 57].

In this work, we consider the honest-majority setting for small number of parties ($n = 3$ and $n = 4$) tolerating at most one malicious corruption ($t = 1$). Next, we outline the relevant literature related to MPC with small number of parties beyond the two-party case.

## 1.1 Related Work

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

In the domain of constant-round protocols which is the focus of this paper, [51] presents a 3-round efficient 3-party (3PC) protocol tolerating at most one malicious corruption and involving transmission and evaluation of a single garbled circuit. Concurrently, in the 3-party setting, [40] achieves a 2-round protocol whose cost is essentially that of 3 garbled circuits. However, both these protocols achieve a weaker notion of security i.e security with selective abort (corrupt party can selectively deprive some of the honest parties of the output). In the presence of a broadcast channel, the 3PC of [51] can additionally achieve unanimous abort (where either all or none of the honest parties output abort), albeit for specific class of functions that give same output to all. The work of [40] presents a 2-round 4-party (4PC) protocol tolerating single corruption that achieves guaranteed output delivery in the absence of broadcast channel. Since the focus of [40] is on minimizing the number of rounds of interaction, the protocol comprises of several parallel instances of private simultaneous message (PSMs) which when instantiated with garbled circuit (GC) would sum upto communication of 12 GCs. The recent work of [54] explores the exact round complexity of 3PC protocols under various security notions including fairness and guaranteed output delivery. While the protocols are round-optimal, they involve a minimum of 3 GCs. The work of [21] explores the case of 5-party with two malicious corruptions and relies on distributed garbling approach of [11] (which is more expensive than Yao's garbling). Recent paper of [12], improving on the distributed garbling techniques of [11], proposes an honest majority protocol with $n > 3t$ and shows practical implementation for 31 parties. The results mentioned above are designed in the honest majority. [23] studies 3PC in dishonest majority setting. In summary, the most relevant work that is close to our work efficiency-wise is that of [51] which we compare with.

There have been a flurry of works in the high-throughput domain recently [2–4, 32]. In 3-party setting, [4] and [32, 38] present semi-honest and maliciously secure protocols respectively that are extremely fast on standard hardware. [2] significantly improves over the protocol of [32], achieving the computation rate of 1.15 billion AND gates/second. In the 4 party setting, the work of [38] provides a construction that is secure against one malicious corruption based on the dual execution approach. They incur communication of 1.5 bits per party per gate for boolean circuits and thus offer a performance that is 4.5 times better than that of [2]. [38] also includes protocol variants for achieving fairness and robustness.

## 1.2 Our Contribution

In this paper, we present efficient constant-round constructions of 3PC and 4PC achieving strong security notions of fairness and guaranteed output delivery that tolerate one active corruption. Our constructions, all based on symmetric-key primitives are built from GC. We outline our results below. For empirical purpose, the circuits of AES-128, SHA-256 and MD5 are used as benchmarks.

*3PC with fairness.* In the minimal network setting of pairwise-private channels, our 3PC protocol with fairness consumes four rounds and involves transmission and evaluation of a *single* GC. Our protocol shows a minimal overhead of 0.06–0.16 ms, 0.03–0.8 ms, 0.21–0.5 s and 5.63–10.74 KB over the 3PC of [51], in terms of the average computation time, LAN runtime, WAN runtime and communication, where average is taken over the number of parties and the range is taken over the choice of benchmark circuits. The nominal overhead to trade fairness over abort security makes our construction a better choice for practical purposes. This protocol has a natural extension to more than 3 parties (still for one corruption) with neither inflating the round complexity nor the number of GCs.

On the technical side, constructing on the ideas of [40, 51], our protocol has two parties enacting the role of the garblers and the remaining party acting as an evaluator for a GC. Contrary to [40, 51], we use a different kind of garbling scheme, namely an oblivious one that hides the circuit output when the decoding information is withheld, in order to enforce fairness. Specifically, an evaluator gets the decoding information only when it shares the result of the GC evaluation to its garblers. Crucial to enforce unanimity across the honest parties, we introduce a clever 'proof mechanism' that allows a garbler to convince its co-garbler about the correctness of the final output. When the evaluator is corrupt and aids only one of its garblers to obtain the output, the technique allows both the honest parties to conclude the same output.

*4PC with guaranteed output delivery.* In the 4-party setting, we present an efficient protocol that achieves guaranteed output delivery in five rounds, assuming just pairwise-private channels. Our

protocol involves communication of a *single* GC compared to the 2-round protocol of [40] that incurs a cost of 12 GCs. Our protocol has asymmetric roles for each party involved and as a result, interestingly, our protocol gives better performance compared to the 3PC of [51]. The protocol terminates in three rounds when no malicious behaviour takes place and has minimal communication (and negligible computation) done in last two rounds. We take readings for both 3-round run and 5-round run of the protocol. For the former, our protocol shows a *gain* of 0.19–2.61 ms, 0.17–2.45 ms and 18.63–500.56 KB respectively compared to the 3PC of [51] in terms of average computation time, LAN runtime and communication. The overhead for WAN runtime is minimal and amounts to 0.02–0.31 s. When the protocol is stretched to 5 rounds, the gains reported above remain unaffected (or witness negligible decrease). In terms of average WAN runtime, the overhead increases to 0.51 – 0.83 s, reflecting the increase in round complexity. At the expense of one extra GC, we also present a 4-round 4PC primarily as a theoretical contribution, which also terminates in three rounds when no malicious behaviour takes place.

On the technical side, deviating from the usual approach of appointing $(n-1)$ garblers and one evaluator in the GC-based protocols of $n$ parties [21, 40, 51], we explore the setting of two garblers and one evaluator in our 5-round 4PC. The fourth party participates only to share its input among the rest. At the heart of this protocol, lie a few neat tricks. We overlap the input commitment phase with evaluation of GC, yet ensure computation on committed input alone by a convenient and clever use of oblivious garbling scheme. It further banks on a technique of identifying a trusted (honest) party amongst the participants and then enabling her to evaluate the function on clear. Specifically, the protocol looks for potential misbehaviour, identifies conflict and assigns the left-out third party the role of trusted-third party (TTP). The TTP, on receiving the view of any other party, takes charge of computing the function on direct inputs and forwarding the output to all. To bring down the round complexity to four, we present a protocol that explores the setting of two garblers and two evaluators and leverages the guarantee of having at least one honest evaluator.

*3PC with guaranteed output delivery.* With an additional broadcast channel, we present a 5-round 3PC protocols with guaranteed output delivery at the cost of communication of a *single* GC. A broadcast channel is inevitable in this regime owing to the results of [25]. We ensure that the broadcast communication is nominal and most importantly, independent of the circuit size. Our implementation, using a physical UDP broadcast channel available on LAN, shows that the average computation time, LAN runtime and communication overhead are 0.16–0.3 ms, 1.52–3 ms and 0.19–0.46 KB respectively over that of [51]. For the worst case run when the execution is stretched to 5 rounds, there is negligible change in the computation and LAN runtime, but communication overhead is witnessed to increase to a value between 0.21–0.57 KB. We do not implement the protocol in WAN as it would require an implementation of a robust broadcast protocol. When the adversary remains semi-honest, this protocol too terminates in 3 rounds and the extra communication and computation needed in the last two rounds is almost nothing.

On the technical side, our construction embarks on similar idea as that of our fair 3PC protocol, except that a garbling scheme without obliviousness works for us. Similar to our 4PC, this protocol also leverages a clever (yet, different from the one used in 4PC) way of identifying a TTP in the face of misbehaviour and evaluating the function on clear.

*Theoretical and Empirical Comparison.* Our protocols put in the context of the relevant state-of-the-art protocols in terms of number of GCs, rounds and security are given below. 'god' implies guaranteed output delivery.

| Ref. | # Parties | # GCs | Rounds | Security | Broadcast |
|---|---|---|---|---|---|
| [51] | 3 | 1 | 3 | selective abort | ✗ |
| This paper | 3 | 1 | 4 | fairness | ✗ |
| This paper | 3 | 1 | 5 | god | ✓ [25] |
| [40] | 4 | 12 | 2 | god | ✗ |
| This paper | 4 | 2 | 4 | god | ✗ |
| This paper | 4 | 1 | 5 | god | ✗ |

While elaborate experimental results appear later in the paper, we summarize the overhead or gain (indicated by **g**) of our protocols compared to the 3PC of [51] in terms of *average* computation time, LAN runtime, WAN runtime and communication cost, where the average is taken over the number of parties and the range is taken over the choice of circuits. We show in bracket the increase in the overhead or decrease in the gain for the worst case 5-round run of our 3PC and 4PC with guaranteed output delivery. With respect to our 4-round 4PC with guaranteed output delivery, in the worst case run, we save one round at the expense of one garbled circuit over our 5-round 4PC which amounts to a value in the range 72 KB – 1530 KB for the benchmark circuits.

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

*Roadmap:* Our needed primitives appear in Section 2. Our efficient 3PC protocols achieving fairness and guaranteed output delivery are presented in Section 3 and 6 respectively. The 4PC protocol with rounds 5 and 4 appear in Section 4 and 5 respectively. The experimental results are presented in Section 7. The security model appears in Appendix A. The complete security proofs appear in the full version of our paper [19].

## 2 PRELIMINARIES

### 2.1 Model and Notations

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

We denote the computational security parameter by $\kappa$. A negligible function in $\kappa$ is denoted by $\mathtt{negl}(\kappa)$. A function $\mathtt{negl}(\cdot)$ is *negligible* if for every polynomial $p(\cdot)$ there exists a value $N$ such

that for all $m > N$ it holds that $\mathtt{negl}(m) < \frac{1}{p(m)}$. We denote by $[x]$, the set of elements $\{1, \ldots, x\}$ and by $[x, y]$, the set of elements $\{x, x+1, \ldots, y\}$ such that $y > x$. For any $x \in_R \{0,1\}^m$, $x^i$ denotes the $i$th bit of $x$, $i \in [m]$. We use $\|_{i\in[n]} x_i$ to denote concatenation of strings $x_i$. Let $S$ be an infinite set and $X = \{X_s\}_{s\in S}, Y = \{Y_s\}_{s\in S}$ be the 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 $p(\cdot)$.

We prove the security of our protocols in the standard real/ideal world paradigm. The security definition and the required functionalities are given in Appendix A.

## 2.2 Primitives

*Garbling Schemes.* 'Garbling Schemes' traditionally used as a technique in secure protocols, were formalized as a primitive by Bellare et al. [10] and were assigned well-defined notions of security, namely *correctness*, *privacy*, *obliviousness*, and *authenticity*. This terminology has largely been adopted by works that followed in this domain [39, 43, 58].

A garbling scheme $\mathcal{G}$ is characterized by a tuple of PPT algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ described below. With the exception of Gb, are all deterministic.

- $\mathsf{Gb}\,(1^\kappa, C)$ 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 an encoded input $\mathbf{X}$.
- $\mathsf{Ev}\,(\mathbf{C}, \mathbf{X})$ evaluates $\mathbf{C}$ on $\mathbf{X}$ to produce an 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 deferred to Appendix B.1. We are interested in a class of garbling schemes referred to as *projective* in [10]. 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 $\mathbf{X}$ corresponding to $x = (x_i)_{i\in[n]}$ can be interpreted as $\mathbf{X} = \mathsf{En}(x, e) = \left(e_i^{x_i}\right)_{i\in[n]}$.

Our 3PC with fairness and 4PC with guaranteed output delivery protocol rely on garbling schemes that are simultaneously correct, private, oblivious and authentic. Our 3PC protocol with guaranteed output delivery relies on garbling schemes that are correct, private and authentic. It further needs an additional decoding mechanism denoted as *soft decoding* algorithm sDe [51] 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 (Com, Open) defined as follows. Given a security parameter $\kappa$, a common parameter pp, message $x$ and random coins $r$, PPT algorithm Com outputs commitment $c$ and corresponding opening information $o$. Given $\kappa$, 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 hiding (a commitment must not leak information about the underlying message) properties. For our 3-party protocols, binding is required to hold only with respect to uniformly chosen public parameter pp, while our 4-party protocols demand stronger form of binding that must hold even against adversarially chosen public parameter. There exist instantiations based on one-way functions for the former and injective one-way function (alternately one-way permutation) for the latter. We denote the NICOM with the stronger binding property as strong NICOM that consists of (sCom, sOpen). The formal definitions of the properties and the instantiations of NICOM based on symmetric key primitives are given in Appendix B.2.

We also need a NICOM scheme that admits equivocation property for our fair 3PC. An equivocal NICOM (eNICOM) is a NICOM that allows equivocation of a certain commitment to any given message with the help of a trapdoor. Apart from the usual two algorithms (eCom, eOpen), eNICOM comprises of:

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

An eNICOM satisfies correctness, hiding and binding properties much like the NICOM does. The hiding property of eNICOM is slightly changed compared to that of NICOM taking the equivocation property into account. This new definition implies the usual hiding definition. In our fair protocol, the public parameter epp for eNICOM is generated jointly by two parties acting garblers so that the trapdoor $t$ remains distributed among them. Our instantiations are based on Naor [52] and programmable random oracle, all of which admit the above property. We thus rewrite eGen in our fair protocol as $(\mathsf{epp}, t_1, t_2) \leftarrow \mathsf{eGen}(1^\kappa)$, where $t_i$ denotes the share of trapdoor held by garbler $P_i$ ($i \in [2]$). Both these shares are necessary to perform equivocation. The formal definitions and instantiations appear in Appendix B.3.

In the implementation of our protocols, we use the random oracle based construction for all the above variants of NICOM which is implemented using SHA-256 .

*Replicated Secret Sharing (RSS) [27, 42].* We use a 3-party replicated secret sharing scheme private against one corruption (1-private). Informally, for a secret $s$ to be shared over a boolean

field $\mathbb{F}_2$, we randomly choose $r_1$, $r_2$ and compute $r_3$ such that $s = r_1 \oplus r_2 \oplus r_3$ (where $r_3 = s \oplus r_1 \oplus r_2$). We refer to $r_1, r_2, r_3$ as the three shares of $s$. Each of the 3 participating parties say $P_1$, $P_2$, $P_3$ are given access to two among the three shares i.e $(r_2, r_3)$, $(r_1, r_3)$ and $(r_1, r_2)$ respectively. Reconstruction of $s$ is possible by combining the shares held by any two among the three parties. However, given only the shares of a single party, the distribution of shares appears random and hence $s$ remains private. We say that two parties say $P_1, P_3$ hold consistent shares if $r_2' = r_2$ where $(r_2', r_3)$ are the shares held by $P_1$ and $(r_1, r_2)$ are the shares held by $P_3$ [40].

*Pre-Image Resistance Hash [56].* Consider a hash function family $\mathsf{H}: \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{Y}$. The hash function $\mathsf{H}$ is said to be pre-image resistant if for all probabilistic polynomial-time adversaries $\mathcal{A}$, given $y = \mathsf{H}_k(x)$ where $k \in_R \mathcal{K}; x \in_R \{0,1\}^m$, $\Pr[x' \leftarrow \mathcal{A}(k, y) : \mathsf{H}_K(x') = y]$ is *negligible* in $\kappa$, where $m = \text{poly}(\kappa)$.

## 3 3PC WITH FAIRNESS

In this section, we present an efficient fair 3PC protocol that consumes 4 rounds in a network constituting of only pairwise-private channels. The starting point of our protocol is that of [51]. In the protocol of [51], $P_1, P_2$ act as garblers while $P_3$ acts as an evaluator. The garblers use common randomness to construct the same GC individually. Since at most one party can be corrupt, a comparison of GCs received from the garblers allows the evaluator $P_3$ to conclude its correctness. Besides, $P_3$ additively shares his input among the garblers at the beginning of the protocol. This eliminates the need of oblivious transfer (OT) to transfer the evaluator's encoded input, as the garblers can directly send the encoded inputs corresponding to their own input as well as the share of $P_3$'s input held by them. To force the garblers to input encoded inputs (the keys) that are consistent with the GCs, the following technique is adopted. Together with the GC, each garbler also generates the commitment to the encoding information using the common shared randomness and communicates to the evaluator. Again a simple check on whether the set of commitments are same for both the garblers allows to conclude their correctness. Now it is infeasible for the garblers to decommit the encoded input corresponding to their own input and the evaluator's share to something that are inconsistent to the GC without being caught. Following a common trick to hide the inputs of the garblers, the commitments on the encoding information corresponding to every bit of the garblers' input are sent in permuted order that is private to the garblers. Now if evaluation of the GC by $P_3$ is successful, $P_3$ computes the output using soft decoding on the encoded output $\mathbf{Y}$. $P_3$ then sends $\mathbf{Y}$ to the garblers, enabling them to decode the output. For a function where all parties receive same output, depending upon whether $\mathbf{Y}$ is broadcast or sent over pairwise channel, the protocol achieves security with abort or selective abort respectively. Specifically, in the latter case when $\mathbf{Y}$ is sent over point-to-point channel, a corrupt $P_3$ may choose to send $\mathbf{Y}$ to only one of the garblers, thereby achieving security with selective abort.

In the protocol of [51], the only scenario in which fairness is violated is when a malicious $P_3$ computes the output via soft decoding but chooses not to send (or sends wrong) encoded output $\mathbf{Y}$ to the garblers. At a high-level, we overcome this limitation by using *oblivious* garbling instead and withholding the decoding information $d$ from $P_3$ until he forwards $\mathbf{Y}$. Obliviousness ensures that $P_3$ gets no information regarding output as long as $d$ is unknown to him. A corrupt $P_3$ is forced to send $\mathbf{Y}$ to the garblers if he wants to learn the output, in which case at least one the garblers $P_1, P_2$ also learn the output. Authenticity ensures that $P_3$ cannot forge an encoded output $\mathbf{Y}' \neq \mathbf{Y}$ such that its decoding is valid. Even if $P_3$ chooses to abort, fairness is achieved as no party learns the output. However, this new step gives rise to the following issues: (a) A corrupt garbler may send incorrect decoding information to an honest $P_3$ who forwarded $\mathbf{Y}$; (b) A corrupt $P_3$ may send the correct encoded output $\mathbf{Y}$ (obtained by GC evaluation) to only one of the garblers. To tackle (a), the garblers are made to commit to the decoding information which $P_3$ can verify by means of cross-checking across garblers. The binding property of the commitment scheme prevents the corrupt garbler from lying about the decoding information later. The second issue is trivial to resolve with a broadcast channel. Without a broadcast channel, each garbler is made to forward the encoded output received from the evaluator to its co-garbler with a "proof" that he indeed received the encoded output from $P_3$. Without a proof, a corrupt garbler may "pretend" to have received the encoded output from honest $P_3$, whereas in reality $P_3$ was unable to evaluate the GC.

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

The protocol f3PC appears in Fig. 1. We use an eNICOM to commit to the decoding information. This is due to a technicality that arises in the security proof explained in the full version [19]. Our proofs and proposed optimizations for f3PC which are incorporated in our implementation are explained subsequently. Lastly, the protocol f3PC cannot be naively extended to obtain guaranteed

**Figure 1: Protocol** f3PC

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

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

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

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

**Round 1:**

- $P_1$ chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends $s$ to $P_2$.
- $P_1$ does the following (Similar steps will be executed by $P_2$): Sample $t_1$ corresponding to its share $\text{epp}_1$ for eNICOM. Compute $h_1 = \mathsf{H}(r_1)$, where $r_1$ is chosen uniformly at random. Send $\{\text{epp}_1, h_1\}$ to $P_2$ and $\{h_1, r_1\}$ to $P_3$.
- $P_3$ samples pp for the NICOM and sends $(x_{31}, \text{pp})$ to $P_1$, $(x_{32}, \text{pp})$ to $P_2$.

**Round 2:**

- $P_i (i \in [2])$ does the following:
  - Compute epp using $\text{epp}_i$ and the share $\text{epp}_j$ received from $P_j$ $(j \in [2] \setminus i)$. Forward $h_j$ received from $P_j$ to $P_3$.
  - Compute GC $(\mathbf{C}, e, d) \leftarrow \text{Gb}(1^\kappa, C)$ using randomness from $G(s)$. Assume $\{e^0_\alpha, e^1_\alpha\}_{\alpha\in[\ell]}$, $\{e^0_{\ell+\alpha}, e^1_{\ell+\alpha}\}_{\alpha\in[\ell]}$, $\{e^0_{2\ell+\alpha}, e^1_{2\ell+\alpha}\}_{\alpha\in[2\ell]}$ correspond to the encoding information for the input of $P_1, P_2$ and shares of $P_3$ respectively.
  - Choose permutation strings $p_1, p_2 \in_R \{0,1\}^\ell$ for the garblers' input wires and generate commitments to $e$ and $d$ using randomness from $G(s)$. For $b \in \{0,1\}$, $(c^b_\alpha, o^b_\alpha) \leftarrow \text{Com}(\text{pp}, e^{p_1^\alpha \oplus b}_\alpha)$, $(c^b_{\ell+\alpha}, o^b_{\ell+\alpha}) \leftarrow \text{Com}(\text{pp}, e^{p_2^\alpha \oplus b}_{\ell+\alpha})$ for $\alpha \in [\ell]$ and $(c^b_{2\ell+\alpha}, o^b_{2\ell+\alpha}) \leftarrow \text{Com}(\text{pp}, e^b_{2\ell+\alpha})$ for $\alpha \in [2\ell]$. Let $(c, o) \leftarrow \text{eCom}(\text{epp}, d)$. Set $\mathcal{B} = \left\{\text{epp}, \mathbf{C}, \{c^b_\alpha\}_{\alpha\in[4\ell], b\in\{0,1\}}, c\right\}$.
- $P_1$ computes $m_1 = x_1 \oplus p_1$ and sends to $P_3$: $\mathcal{B}$, the openings of the commitments corresponding to $(x_1, x_{31})$ i.e $\{o^{m_1^\alpha}_\alpha, o^{x_{31}^\alpha}_{2\ell+\alpha}\}_{\alpha\in[\ell]}$ and $m_1$. Similarly, $P_2$ computes $m_2 = x_2 \oplus p_2$ and sends to $P_3$: $\mathcal{B}$, the openings of the commitments corresponding to $(x_2, x_{32})$ i.e $\{o^{m_2^\alpha}_{\ell+\alpha}, o^{x_{32}^\alpha}_{3\ell+\alpha}\}_{\alpha\in[\ell]}$ and $m_2$.
- $P_3$ does the following computation locally.
  - Abort if $\mathcal{B}$ or $(h_1, h_2)$ received from $P_1, P_2$ is not identical or $\mathsf{H}(r_i) \neq h_i$ for some $i \in [2]$.
  - Abort if $(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_{31}, \mathbf{X}_{32})$ contains $\perp$ where for $\alpha \in [\ell]$ : $\mathbf{X}^\alpha_1 = \text{Open}(\text{pp}, c^{m_1^\alpha}_\alpha, o^{m_1^\alpha}_\alpha)$, $\mathbf{X}^\alpha_2 = \text{Open}(\text{pp}, c^{m_2^\alpha}_{\ell+\alpha}, o^{m_2^\alpha}_{\ell+\alpha})$, $\mathbf{X}^\alpha_{31} = \text{Open}(\text{pp}, c^{x_{31}^\alpha}_{2\ell+\alpha}, o^{x_{31}^\alpha}_{2\ell+\alpha})$, $\mathbf{X}^\alpha_{32} = \text{Open}(\text{pp}, c^{x_{32}^\alpha}_{3\ell+\alpha}, o^{x_{32}^\alpha}_{3\ell+\alpha})$.
  - Else set $\mathbf{X} = \mathbf{X}_1|\mathbf{X}_2|\mathbf{X}_{31}|\mathbf{X}_{32}$ and run $\mathbf{Y} \leftarrow \text{Ev}(\mathbf{C}, \mathbf{X})$ for $\mathbf{C} \in \mathcal{B}$.

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

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

The parties do the following.

- $P_3$ runs $d \leftarrow \text{eOpen}(\text{epp}, c, o)$ where $P_3$ received $o$ from $P_i$ $(i \in [2])$. For $d \neq \perp$, $P_3$ outputs $y \leftarrow \text{De}(\mathbf{Y}, d)$.
- $P_i$ $(i \in [2])$ does the following if $y = \perp$: If received $(y', r'_i)$ from $P_{[2]\setminus i}$ such that $\mathsf{H}(r'_i) = h_i$, set $y = y'$.

output delivery even in the presence of a broadcast channel (which is necessary due to [25]). When the evaluator fails to obtain the encoded output, there should be a way to compute the output which either seems to need more parties to enact the role of the evaluator and consequently involvement of more than one GCs or seems to require more than four rounds. We take the latter way-out and design a 5-round protocol in Section 6.

## 3.1 Correctness and Security

Theorem 3.1. *The protocol* f3PC *is correct.*

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

While the formal proof appears in the full version [19], we give intuition for fairness and state the theorem below. We need to argue that a corrupt party gets the output of the computation if and only if the honest parties receive the output. For the forward direction assume that a corrupt party gets the output. Say the evaluator $P_3$ is corrupt. Due to oblivious garbling, $P_3$ would obtain the output only if given access to decoding information. This would occur only if he had sent a valid $(\mathbf{Y}, r_j)$ to at least one of the garblers say $P_i$ ($P_j$ is the co-garbler) i.e., $\text{De}(\mathbf{Y}, d) \neq \perp$ and $\mathsf{H}(r_j) = h_j$. $P_i$ would communicate $(\mathbf{Y}, r_j)$ to $P_j$ as well which would be verified

and subsequently accepted by $P_j$. Thus all parties would learn the output. The case of corrupt garbler, say $P_1$ obtaining the output is straightforward - it would occur only in the case when the honest $P_3$ is able to evaluate the garbled circuit successfully. In this case, it is easy to see that the honest garbler $P_2$ and evaluator $P_3$ would be able to obtain the output using encoded output and decoding information received from the other respectively.

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

Theorem 3.2. *If one-way functions exists, then protocol* f3PC *securely realizes the functionality $\mathcal{F}_{\mathsf{Fair}}$ (Fig. 8) against a malicious adversary that corrupts at most one party.*

## 3.2 Optimizations and generalization

We propose the following optimizations to improve communication efficiency. Firstly, $P_1$ and $P_2$ treat the common message $\mathcal{B}$ sent privately to $P_3$ in Round 2 as a string $\mathcal{B}$, divided into equal halves $\mathcal{B} = \mathcal{B}^1||\mathcal{B}^2$. $P_1$ sends $\mathcal{B}^1$ and $\mathsf{H}'(\mathcal{B}^2)$ while $P_2$ sends $\mathsf{H}'(\mathcal{B}^1)$ and $\mathcal{B}^2$ to $P_3$, where $\mathsf{H}'$ refers to a collision-resistant hash function. This would suffice for $P_3$ to verify if $P_1, P_2$ agree on a common $\mathcal{B}$. This optimization technique not only reduces the communication, but also improves the latency (transmission time) when both $P_1, P_2$ run at the same time [51]. The second optimization is to use equivocal commitment on the hash of the decoding information (collision-resistant hash), rather than simply committing on the decoding information.

Our protocol design has a natural extension to more than 3 parties (still for one corruption) without inflating the round complexity and number of GCs. The generalized protocol comprises of $(n-1)$ garblers who use common randomness for garbling and a single evaluator who additively shares her input amongst the garblers. For $n > 3$, the correctness of GC can be concluded based on majority rule on the GCs received from the garblers.

# 4 4PC WITH GUARANTEED OUTPUT DELIVERY

In this section, we propose an efficient 5-round 4PC secure against one active corruption, assuming pairwise channels. Our protocol involves communication and computation of just one GC, in contrast to the protocol of [40] that requires 12 GCs. We take the route of employing two garblers and one evaluator as in our fair 3PC protocol. The fourth party simply shares its input amongst the rest. When the evaluator is honest, our protocol ensures that either an honest party identifies the corrupt party or a conflict (assured to include the corrupt party), or the honest evaluator is successful in GC evaluation by the end of Round 2. In the former case, the honest party would identify at least one honest party, to whom she sends her possessed input shares in Round 3. We use replicated secret sharing (RSS) that allows reconstruction of the output based on views of any two (honest) parties. In the latter scenario, the encoded output obtained upon GC evaluation is instantly used for output computation by all the parties in Round 3. Thus, in either scenario, at least one of the honest parties will be able to compute the output latest by Round 3 and everyone will receive it by Round 4. On the other hand, a corrupt evaluator can drag the honest parties up to Round 4 to reveal its identity. This is the only case that makes our protocol run for 5 rounds where the last round is used by the honest parties to exchange their possessed shares to compute the output on clear.

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

## 4.1 Protocol for Input Consistency

Our protocol $\mathsf{InputCommit}_i$ that runs for two rounds, enforces input consistency of party $P_i$'s secret $x_i$ as per RSS. Recall that as per RSS for three shareholders, $P_i$ makes three shares of its secret $x_i$ as $x_i = \oplus_{P_j \in \mathcal{P}_i} x_{ij}$ where $\mathcal{P}_i = [4] \setminus i$ denotes the shareholders (i.e. all but $P_i$). The share $x_{ij}$ goes to all but $P_i$ and $P_j$, namely to the set of parties in $\mathcal{P}_{ij} = \mathcal{P} \setminus \{P_i, P_j\}$. Now to ensure that a corrupt $P_i$ remains committed to its secret or a corrupt shareholder $P_j$ later cannot open a share of $P_i$ differently, we use commitments on the shares. Namely, in the first round, commitments on input shares are distributed by $P_i$ to all while the openings are sent only to the relevant shareholders. In the second round, the shareholders exchange the commitments received in the first round, while the openings are exchanged only with the relevant shareholders. A simple majority rule suffices to conclude on the commitment $\mathsf{c}_{ij}$ of the 'committed' share $x_{ij}$. When no honest majority is found, it can be concluded that $P_i$ is corrupt and his input is taken as a default value by all parties. When the commitment and the opening distributed by $P_i$ is found to be inconsistent, then $P_i$ is identified as corrupt. When the commitment as distributed by $P_i$ and forwarded by $P_j$ contradict, then $P_i$ and $P_j$ are put in conflict set.

A share $x_{ij}$ is said to be 'committed' if each honest $P_\alpha \in \mathcal{P}_i$ holds $\mathsf{c}_{ij}$ and each honest $P_\beta \in \mathcal{P}_{ij}$ holds $\mathsf{o}_{ij}$ such that $\mathsf{c}_{ij}$ opens to $x_{ij}$ via $\mathsf{o}_{ij}$. A secret $x_i$ is said to be 'committed' if each of its three shares are committed. An honest party always 'commits' to its secret. When a corrupt party does not commit to a secret, it is either identified as corrupt or found to be in conflict by at least one honest party. For the commitments, we use a strong NICOM according to which binding holds even for adversarially chosen public parameter of the NICOM (see Appendix B.2). Looking ahead the strong NICOM ensures that $P_i$ itself cannot change its committed secret later and also cannot keep two different parties on different pages in terms of the opening information $\mathsf{o}_{ij}$. Protocol $\mathsf{InputCommit}_i$ appears in Fig. 2.

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

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

**Figure 2: Protocol** $\mathsf{InputCommit}_i()$

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

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

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

**Primitives:** A NICOM (sCom, sOpen) with strong binding property (Appendix B.2), a 3-party 1-private RSS.

**Round 1:**

- $P_i$ shares his input as $x_i = \oplus_{j\in\mathsf{ind}(\mathcal{P}_i)} x_{ij}$.
- $P_i$ samples $\mathsf{pp}_i$ and generates commitments on shares $x_{ij}$ for $j \in \mathsf{ind}(\mathcal{P}_i)$ as $(\mathsf{c}_{ij}, \mathsf{o}_{ij}) \leftarrow \mathsf{sCom}(\mathsf{pp}_i, x_{ij})$
- For every $x_{ij}$, $P_i$ sends $(\mathsf{pp}_i, \mathsf{c}_{ij})$ to $\mathcal{P}_i$ and $\mathsf{o}_{ij}$ to $\mathcal{P}_{ij}$.

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

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

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

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

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

*Proof.* For the $j$th ($j \in \mathsf{ind}(\mathcal{P}_i)$) share of $P_i$, there are two cases based on whether $P_i$ sends the same common message $(\mathsf{pp}_i, \mathsf{c}_{ij})$ to at least two among the parties in $\mathcal{P}_i$ with valid corresponding opening $\mathsf{o}_{ij}$ sent to every party in $\mathcal{P}_{ij}$. If not, the exchange of messages among the honest parties in Round 2 will not constitute a majority and all the honest parties would detect $P_i$ to be corrupt and a default value will be taken as $x_{ij}$. Else, $\mathsf{c}_{ij}$ would be accepted as the commitment for the $j$th share. The exchange of opening $\mathsf{o}_{ij}$ among the parties in $\mathcal{P}_{ij}$ ensure that they have access to the corresponding unique committed share $x_{ij}$. The uniqueness of the share is ensured by the binding property of commitment scheme. □

## 4.2 Our protocol

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

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

The repetitive disbursal of shares in RSS brings along another issue. Both the garblers possess the share $x_{34}$. An evaluator receives encoded input for these shares from both the garblers, as per the protocol. A corrupt evaluator $P_3$ can exploit this step to obtain encoded inputs for two different versions of the share $x_{34}$ (by dealing to the garblers) and subsequently evaluates the circuit on multiple inputs. While having the decoding information hidden would not leak the clear outputs, the corrupt evaluator, on holding the

**Figure 3: Protocol** g4PC()

**Inputs:** Party $P_\alpha$ has $x_\alpha$ for $\alpha \in [4]$.

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

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

**Primitives:** $\mathcal{G}$ = (Gb, En, Ev, De) that is correct, private, oblivious and authentic, a NICOM (Com, Open), a PRG G, a preimage-resistant Hash H and sub-protocol $\mathsf{InputCommit}_\alpha$ (Figure 2) for every $P_\alpha \in \mathcal{P}$.

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

- $P_1$ chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends $s$ to $P_2$.
- $P_3$ samples $\mathsf{pp}_3$ for NICOM and sends to $P_1, P_2$.

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

- $P_g (g \in [2])$ locally computes the following:
  - Compute garbled circuit $(\mathbf{C}, e, d) \leftarrow \mathrm{Gb}(1^\kappa, C)$ using randomness from $\mathrm{G}(s)$. Assume $\{e^0_\alpha, e^1_\alpha\}_{\alpha\in[3\ell]}$, $\{e^0_{3\ell+\alpha}, e^1_{3\ell+\alpha}\}_{\alpha\in[3\ell]}$, $\{e^0_{6\ell+\alpha}, e^1_{6\ell+\alpha}\}_{\alpha\in[3\ell]}$, $\{e^0_{9\ell+\alpha}, e^1_{9\ell+\alpha}\}_{\alpha\in[3\ell]}$ correspond to the encoding information for the input shares of $P_1, P_2, P_3, P_4$ respectively (w.l.o.g).
  - Let $p_{ij} \in_R \{0,1\}^\ell$ be permutation string for input wires derived from randomness $\mathrm{G}(s)$ corresponding to $P_i$'s shares i.e $\{x_{ij}\}_{j\in\mathsf{ind}(\mathcal{P}_i)}$ for $i \in [4]$ and $p_i \leftarrow \|_{j\in\mathsf{ind}(\mathcal{P}_i)} p_{ij}$.
  - Generate commitments to $e$ and $d$ using randomness from $\mathrm{G}(s)$. For $b \in \{0,1\}$ and $\alpha \in [3\ell]$, compute $(c^b_\alpha, o^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}_3, e^{p^\alpha_1 \oplus b}_\alpha)$, $(c^b_{3\ell+\alpha}, o^b_{3\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, e^{p^\alpha_2 \oplus b}_{3\ell+\alpha})$, $(c^b_{6\ell+\alpha}, o^b_{6\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, e^{p^\alpha_3 \oplus b}_{6\ell+\alpha})$, $(c^b_{9\ell+\alpha}, o^b_{9\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}_3, e^{p^\alpha_4 \oplus b}_{9\ell+\alpha})$. Let $(c, o) \leftarrow \mathsf{Com}(\mathsf{pp}_3, \mathsf{H}(d))$. Set $\mathcal{B} = \left\{\mathbf{C}, \{c^b_\alpha\}_{\alpha\in[12\ell], b\in\{0,1\}}, c, \{p_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i3})}\right\}$, where $\{p_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i3})}$ refer to the permutation strings of wires corresponding to the shares known to $P_3$.
- $P_g (g \in [2])$ sends $\mathcal{B}$ to $P_3$ and $c$ to $P_4$. If $\mathcal{C}_g = \emptyset$, $P_g$ sends the openings of the commitments in $\mathcal{B}$ corresponding to $\{x_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{ig})}$ i.e the input shares that it holds at end of **Round 1** and $\mathsf{M}_g = \{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{ig})}$ where $m_{\alpha\beta} = p_{\alpha\beta} \oplus x_{\alpha\beta}$. The common shares, however, are opened by one garbler. The openings corresponding to commitment of $\{x_{13}, x_{14}, x_{34}\}$ are sent only by $P_1$. The openings corresponding to commitment of $\{x_{23}, x_{24}, x_{43}\}$ are sent only by $P_2$.
- $P_3$ locally does
  - Add $\{P_1, P_2\}$ to $\mathcal{F}_3$ if $\mathcal{B}$ received from $P_1, P_2$ is not identical.
  - If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$ (indicating no conflict with the garblers so far), then (a) add $P_g$ to $\mathcal{C}_3$ $(g \in [2])$ when the indices $\{\tilde{m}_{ij} = p_{ij} \oplus x_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i3})}$, computed using its version of $x_{ij}$ and $p_{ij}$, received from $P_g$, mismatches with $\{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i3})}$ received from $P_g$; (b) add $(P_1, P_2)$ to $\mathcal{F}_3$ when $\mathsf{M}_1, \mathsf{M}_2$ received from them is not consistent w.r.t. $\{m_{13}, m_{14}, m_{23}, m_{24}, m_{34}, m_{43}\}$.
  - If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, then add $P_g$ to $\mathcal{C}_3$ when any of the openings sent by $P_g$ $(g \in [2])$ results to $\perp$. Otherwise, it sets $\mathbf{X} = \|_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_i)} \mathbf{X}_{ij}$, where $\mathbf{X}_{ij}$ contains encoded input for $x_{ij}$ and computes $\mathbf{Y} \leftarrow \mathsf{Ev}(\mathbf{C}, \mathbf{X})$ with $\mathbf{C} \in \mathcal{B}$.
- $P_4$ locally adds $\{P_1, P_2\}$ to $\mathcal{F}_4$ if $c$ received from them do not match.

**Round 3:**

- If $\mathcal{C}_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, $P_\alpha$ $(\alpha \in [4])$ sends $\mathsf{O}_\alpha = \{\mathsf{o}_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i\alpha})}$ to $P_\beta$ where $P_\beta \notin \mathcal{C}_\alpha \cup \mathcal{F}_\alpha$ and $(\mathsf{TTP}, \beta)$ to all.
- If $\mathcal{C}_g = \mathcal{F}_g = \emptyset$, $P_g$ $(g \in [2])$ sends $o$ to $P_3, P_4$.
- If $\mathcal{C}_3 = \mathcal{F}_3 = \emptyset$, $P_3$ sends $\mathbf{Y}$ to $P_1, P_2$ and $P_4$.
- If $P_\alpha$ $(\alpha \in [4])$ receives $\mathsf{O}_\beta$ from $P_\beta$ in Round 3, it uses $\mathsf{O}_\beta$ to open its missing shares $\{x_{i\alpha}\}_{i\in[4]\setminus\{\alpha\}}$. If one of the opening leads to $\perp$, set $\mathcal{C}_\alpha = P_\beta$. Else compute $y = f(\oplus_{j\in\mathsf{ind}(\mathcal{P}_1)} x_{1j}, \oplus_{\mathsf{ind}(\mathcal{P}_2)} x_{2j}, \oplus_{\mathsf{ind}(\mathcal{P}_3)} x_{3j}, \oplus_{\mathsf{ind}(\mathcal{P}_4)} x_{4j})$.
- If $P_g$ $(g \in [2])$ receives $\mathbf{Y}$ from $P_3$ such that $P_3 \notin \mathcal{C}_g$ and $(P_3, P_1), (P_3, P_2) \notin \mathcal{F}_g$, then compute $y \leftarrow \mathsf{De}(\mathbf{Y}, d)$. If $P_4$ receives $\mathbf{Y}$ as above and $o$ from one of the $P_g$s, it computes $y$ after recovering $\mathsf{H}(d) \leftarrow \mathsf{Open}(\mathsf{pp}, c, o)$. If $P_1/P_2/P_4$ receives invalid $\mathbf{Y}$, they populate their respective corrupt set $\mathcal{C}$ with $P_3$. If $P_3$ receives $o$, then it computes $\mathsf{H}(d)$ and subsequently $y$.

**Round 4:**

- If $P_\alpha$ computed $y$, it sends $(y, \mathsf{TTP})$ when elected as TTP and $y$ otherwise to all and terminates.
- If $(\mathsf{TTP}, \beta)$ is received in Round 3 and $(y, \mathsf{TTP})$ is received from $P_\beta$, a party $P_\alpha$ outputs $y$ and terminates. If only the former condition is true, then $P_\alpha$ identifies the sender of the message $(\mathsf{TTP}, \beta)$ as corrupt.
- If $\mathcal{C}_\alpha \neq \emptyset$ and $y$ is received from a party not in $\mathcal{C}_\alpha$, $P_\alpha$ outputs $y$ and terminate.

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

encoded outputs, can conclude if its two different chosen inputs lead to the same output or not. While the issue is very subtle, the fix is quite easy where only one *pre-determined* garbler is given responsibility of releasing the encoded input for the common share $x_{34}$. In order to avoid repeated disclosing of encoded outputs of the common shares between the garblers, this approach is taken for all the common shares, namely $\{x_{13}, x_{14}, x_{23}, x_{24}, x_{34}, x_{43}\}$. To balance load, we ask $P_1$ to open encoded inputs for $\{x_{13}, x_{14}, x_{34}\}$ and $P_2$ to take care of the rest.

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

## 4.3 Correctness and Security

We prove the correctness via a sequence of lemmas.

Lemma 4.4. *For honest $P_i, P_j$, $P_i \notin C_j$ holds.*

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

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

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

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

*Proof.* An honest $P_3$ evaluates the GC and computes $\mathbf{Y}$ when both $\mathcal{F}_3$ and $C_3$ are empty. This implies that the corrupt party 'commits' to its input in Round 1 of its $\mathsf{InputCommit}$ instance (by Lemma 4.2). We can thus conclude that the honest garbler would possess committed input shares of all parties at the end of Round 1 itself and open the encoded inputs accordingly. A potentially corrupt garbler is forced to send the encoded inputs corresponding to committed inputs. Because– (a) if corrupt garbler tries to open different encoded inputs for the shares known to $P_3$, then he is added to $C_3$; (b) if it tries to open different encoded inputs for the shares not known to $P_3$, then $P_3$ would add the pair of garblers to $\mathcal{F}_3$. Thus, in either case, $P_3$ does not evaluate as at least one among $\mathcal{F}_3, C_3$ is non-empty. □

Lemma 4.7. *If the encoded output $\mathbf{Y}$ of a corrupt evaluator $P_3$ is used for output computation by an honest garbler, then it must correspond to committed inputs of all parties.*

*Proof.* An honest garbler, say $P_g$ releases the opening information $o$ for $\mathsf{H}(d)$ and uses the encoded output $\mathbf{Y}$ (such that $\mathsf{De}(\mathbf{Y}, d) \neq \perp$) received from evaluator $P_3$ to compute output if $P_3 \notin C_g$ and $(P_3, P_1), (P_3, P_2) \notin \mathcal{F}_g$. Lemma 4.2 implies that $P_3$ did not misbehave in $\mathsf{InputCommit}_3$ at all and has committed a unique input in Round 1. This implies that $P_3$ receives encoded inputs for committed shares and authenticity ensures that $\mathbf{Y}$ corresponds to the committed inputs of all the parties. Note that authenticity of the garbling scheme is preserved since $P_3$ receives only the preimage-resistant hash of the decoding information in the form $\mathsf{H}(\mathbf{Y}_0)||\mathsf{H}(\mathbf{Y}_1)$ corresponding to each output wire (enabling $P_3$ to compute the output). Here, $\mathbf{Y}_0, \mathbf{Y}_1$ refer to the labels for values 0 and 1 respectively corresponding to an output wire. □

Lemma 4.8. *Protocol* g4PC *is correct.*

*Proof.* We argue that the output $y$ computed corresponds to unique inputs committed by each $P_i$ ($i \in [4]$) during $\mathsf{InputCommit}_i$. It follows from Lemmas 4.3, 4.1 that a corrupt party is forced to commit to a unique input and the honest parties' inputs are established as the committed inputs with public commitments by the end of parallel executions of InputCommit. According to the protocol, an honest party $P_\alpha$ computes output in one of the following ways: (a) via decoding the encoded output Y; (b) via the $\mathsf{O}_\beta$ received from $P_\beta$ on being elected as TTP; (c) on receiving $y$ from an honest party; (d) on receiving $(y, \mathsf{TTP})$ from $P_\beta$ and $(\mathsf{TTP}, \beta)$ from some other party. In case (a), irrespective of whether $P_3$ is honest or corrupt, correctness follows from Lemma 4.6–4.7. The strong binding property of commitment scheme implies the output computed in case (b) is correct irrespective of whether $P_\beta$ is honest or corrupt. The correctness for case (c) follows from case (a) and the fact that the message was received from an honest party. The last case is argued as follows. The chosen TTP, $P_\beta$, is honest, irrespective of whether the message $(\mathsf{TTP}, \beta)$ is received from a corrupt or an honest party. While the former follows from the fact that a corrupt party does not have a corrupt companion to elect, the latter follows from Lemma 4.4–4.5. Now the correctness follows in case (d) from case (b). □

While the full proof of security appears in the full version [19], we provide intuition for guaranteed output delivery and state the theorem below. If the corrupt party misbehaves in one of the InputCommit instances or while communicating the GC and openings on commitment of input labels (as a garbler in round 2), then an honest party invokes TTP on identifying the corrupt or detecting a conflict in Round 3. All the parties get output in Round 4. Otherwise, if $P_3$ is honest and gives out Y, then all the honest parties compute output by the end of Round 3 itself using hash of the decoding information sent by one of the garblers and Y. A corrupt $P_3$ can neither receive decoding information for his non-committed input nor convince honest parties about the corresponding Y. If Y corresponds to its committed input but it sends it only to some honest party or none, the remaining honest parties will receive output from the honest party who receives Y or through $\mathsf{O}_\beta$s sent by other honest parties in Round 5.

Theorem 4.9. *Assuming one-way permutations, protocol* g4PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{GOD}}$ *(Fig. 9) against a malicious adversary that corrupts at most one party.*

## 4.4 Optimizations

The communication efficiency of our g4PC can be boosted similar to as described for f3PC in Section 3.2.

# 5 4PC WITH GUARANTEED OUTPUT DELIVERY IN FOUR ROUNDS

In this section, we propose an efficient 4-round 4PC protocol secure against one active corruption, assuming pairwise channels. Deviating from the approach of [40, 51] and our proposals for 3PC and 4PC, we explore the setting of multiple evaluators, namely two evaluators and two garblers. With a guarantee of an honest evaluator, this protocol achieves guaranteed output delivery at the expense of communication and computation of two copies of the same GC.

**Figure 4: Protocol** g4PC4()

**Inputs:** Party $P_\alpha$ has $x_\alpha$ for $\alpha \in [4]$.

**Common Inputs:** The circuit $C(x_1, x_2, x_3, x_4)$ that computes $f(x_{12} \oplus x_{13} \oplus x_{14}, x_{21} \oplus x_{23} \oplus x_{24}, x_{31} \oplus x_{32} \oplus x_{34}, x_{41} \oplus x_{42} \oplus x_{43})$ each input, their shares and output are from $\{0,1\}^\ell$. $P_3, P_4$ are the evaluators and $(P_1, P_2)$ are the garblers.

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

**Primitives:** $\mathcal{G}$ = (Gb, En, Ev, De) that is correct, private, oblivious and authentic, a NICOM (Com, Open) a PRG G, preimage-resistant Hash H, a 3-party 1-private RSS and sub-protocol $\mathsf{InputCommit}_\alpha$ (Figure 2) for every $P_\alpha \in \mathcal{P}$.

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

- $P_1$ chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends $s$ to $P_2$.
- $P_\upsilon$ ($\upsilon \in \{3,4\}$) samples $\mathsf{pp}_\upsilon$ for NICOM and sends to $P_1, P_2$.

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

- $P_g$($g \in [2]$) locally computes $\mathcal{B}_3$ exactly the way $\mathcal{B}$ is computed in Protocol g4PC. It also computes $\mathcal{B}_4$ with respect to $\mathsf{pp}_4$ in a similar way.
- $P_g$($g \in [2]$) sends $\mathcal{B}_3$ to $P_3$. If $C_g = \emptyset$, $P_g$ sends the openings of the commitments in $\mathcal{B}_3$ corresponding to $\{x_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{ig})}$ i.e the input shares that it holds at end of **Round 1** and $\mathsf{M}_g = \{m_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{ig})}$ where $m_{\alpha\beta} = p_{\alpha\beta} \oplus x_{\alpha\beta}$. Analogous steps are executed with respect to $P_4$. The common shares, however, are opened by one garbler. The openings corresponding to commitment of $\{x_{13}, x_{14}, x_{34}\}$ are sent only by $P_1$. The openings corresponding to commitment of $\{x_{23}, x_{24}, x_{43}\}$ are sent only by $P_2$.
- $P_\upsilon$ ($\upsilon \in \{3,4\}$)) local computation step is same as that of $P_3$ in g4PC (with respect to $C_\upsilon$ and $\mathcal{F}_\upsilon$).

**Round 3:**

- If $C_\alpha \neq \emptyset \vee \mathcal{F}_\alpha \neq \emptyset$, $P_\alpha$ ($\alpha \in [4]$) sends $\mathsf{O}_\alpha = \{\mathsf{o}_{ij}\}_{i\in[4], j\in\mathsf{ind}(\mathcal{P}_{i\alpha})}$ to $P_\beta$ where $P_\beta \notin C_\alpha \cup \mathcal{F}_\alpha$ and $(\mathsf{TTP}, \beta)$ to all.
- If $C_g = \mathcal{F}_g = \emptyset$, $P_g$ ($g \in [2]$) sends $o$ to $P_3, P_4$.
- If $C_\upsilon = \mathcal{F}_\upsilon = \emptyset$, $P_\upsilon$ ($\upsilon \in \{3,4\}$) sends Y to all.
- If $P_\alpha$ ($\alpha \in [4]$) receives $\mathsf{O}_\beta$ from $P_\beta$ in Round 3, it uses $\mathsf{O}_\beta$ to open its missing shares $\{x_{i\alpha}\}_{i\in[4]\setminus\{\alpha\}}$. If one of the opening leads to $\perp$, set $C_\alpha = P_\beta$. Else compute $y = f(\oplus_{j\in\mathsf{ind}(\mathcal{P}_1)} x_{1j}, \oplus_{\mathsf{ind}(\mathcal{P}_2)} x_{2j}, \oplus_{\mathsf{ind}(\mathcal{P}_3)} x_{3j}, \oplus_{\mathsf{ind}(\mathcal{P}_4)} x_{4j})$.
- If $P_g$ ($g \in [2]$) receives a valid Y from $P_\upsilon$ such that $P_\upsilon \notin C_g$ and $(P_\upsilon, P_1), (P_\upsilon, P_2) \notin \mathcal{F}_g$, then compute $y \leftarrow \mathsf{De}(\mathsf{Y}, d)$. If $P_\upsilon$ receives $o$ from one of the $P_g$s, it computes $y$ after recovering $\mathsf{H}(d) \leftarrow \mathsf{Open}(\mathsf{pp}, c, o)$.

**Round 4:**

- If $P_\alpha$ computed $y$ via being elected as TTP, it sends $(y, \mathsf{TTP})$ to all and terminates.
- If $(\mathsf{TTP}, \beta)$ is received in Round 3 and $(y, \mathsf{TTP})$ is received from $P_\beta$, a party $P_\alpha$ outputs $y$ and terminates.

The protocol ensures that the honest evaluator is either successful in GC evaluation or some honest party identifies a corrupt party or a pair of parties in conflict (assured to include the corrupt party) by the end of Round 2. In the former case, the encoded output obtained upon GC evaluation is used for output computation in Round 3 itself. In the latter case, the honest party, having identified at least one honest party, sends his possessed input shares in Round 3. The use of replicated secret sharing (RSS) allows reconstruction of the output based on views of two honest parties by the end of Round 3. All parties obtain output by the end of Round 4.

A single evaluator and three garblers approach seems to require a minimum of 5 rounds (when the evaluator is corrupt) while requiring the same amount of communication. With the above high level idea, we proceed to present our protocol. We reuse the protocol for input consistency (Fig. 2). Similar to our g4PC protocol, each party $P_i$ ($i \in [4]$) maintains a pair of global sets– a corrupt set $\mathcal{C}_i$ and a conflict set $\mathcal{F}_i$ which respectively hold identities of the party detected to be corrupt and pairs of parties detected to be in conflict.

## 5.1 Our protocol

Without loss of generality, $P_1, P_2$ take the role of garblers and $P_3, P_4$ enact the role of evaluators in our protocol g4PC4. We reuse most of the tricks from our 5-round protocol and leverage the presence of an honest evaluator. Specifically, the corrupt evaluator, unlike in our 5-round protocol, cannot drag all the honest parties all the way to Round 4 for its detection. If everything goes as per the protocol and so no honest party elects a TTP in the end of Round 2, the honest evaluator must be able to compute the encoded output Y by the end of Round 2 and help all to get the output in Round 3. Otherwise, all the parties get output via a TTP by Round 4. The presence of an additional evaluator needs communicating one extra copy of the GC. We present the protocol g4PC4 in Fig 4 and state the theorem below. The proof of correctness and security is similar to g4PC (appears in the full version [19]).

Theorem 5.1. *Assuming one-way permutations, our protocol* g4PC4 *securely realizes the functionality* $\mathcal{F}_{\text{GOD}}$ *(Fig. 9) against a malicious adversary that can corrupt at most one party.*

# 6 3PC WITH GUARANTEED OUTPUT DELIVERY

In this section, we describe our efficient 3PC protocol, g3PC with guaranteed output delivery. This protocol necessarily requires a broadcast channel [25]. In accordance with our goal of computation and communication efficiency, the broadcast communication complexity of our (optimized) protocol is independent of circuit size. In terms of communication over private channels, g3PC involves a single GC and is therefore comparable to [51].

Starting with the protocol of [51], the main idea of our protocol is centered around the following neat trick. In a situation where it is publicly known that a pair of parties is in conflict, it must be the case that one among the two specific parties is corrupt. It follows that the third party is honest and therefore entitled to act as the trusted-third party (TTP). Suppose such a TTP is established during the protocol, the other parties send their inputs on clear to this TTP who computes the function on direct inputs and forwards the output to all. Banking on this intuition, we now proceed to give a high-level description of our protocol.

In the first round, similar to f3PC, $P_3$ shares his input while the garblers agree upon common randomness. In round 2, garblers broadcast the common message computed using shared randomness, namely the GC and commitment on encoding information. Additionally, the garblers privately send the opening of relevant commitments, namely corresponding to their own input and the input share of $P_3$ held by them. If the broadcast messages are identical and openings are valid then $P_3$ can begin evaluating the GC. However, if the broadcast messages mismatch, it can be publicly inferred that $P_1, P_2$ are in conflict and therefore $P_3$ is eligible to enact the role of TTP. We extend this idea to the case when broadcast messages are identical but $P_3$ locally identifies one of the garblers to be corrupt. In this scenario, say $P_3$ identified $P_2$ to be corrupt. Then, $P_3$ makes this conflict public in Round 3 via broadcast. Consequently $P_1$ is entitled to act as the TTP. The protocol ensures that if $P_3$ fails to evaluate the GC, a TTP is established at most by Round 3. If the TTP is established, the parties send their inputs on clear to the TTP in Round 4 who computes and subsequently sends the output to all in the final round of the protocol.

An issue that surfaces in the above approach is that a corrupt $P_3$ who has successfully evaluated the GC with respect to his input $x_3$ shared in the round 1, might pretend to be in conflict with one of the garblers, say $P_2$. Now $P_1$ would be established as the TTP. $P_3$ can now send $x_3' \neq x_3$ to $P_1$ and get the output corresponding to $x_3'$ as well. This violates security since $P_3$ gets outputs corresponding to his two chosen inputs. To handle this, we adopt the following strategy: The evaluator $P_3$ broadcasts the commitment on his shares in Round 1 and sends the openings of shares to the respective garbler. A garbler who receives invalid opening is allowed to publicly raise a conflict with $P_3$ in Round 2, establishing his co-garbler as the TTP. If valid openings are issued, $P_3$ is committed to each of his shares and therefore his input. The binding property of commitment ensures that the TTP computes output with respect to $P_3$'s shares distributed in Round 1. Tying up the loose ends, if $P_3$ is identified to be corrupt by both garblers, then $P_1$ is chosen to be the TTP by default.

In a nutshell, $P_3$ acts as TTP only when common message broadcast by garblers are not identical. Contrarily, a garbler, say $P_1$, is TTP when either $P_3$ locally identified $P_2$ to be corrupt at the end of Round 2 (due to invalid opening of commitment on encoded inputs) or $P_2$ found $P_3$ to be corrupt at the end of Round 1 (inconsistent opening of commitment of $P_3$'s input share sent to $P_2$). Also, $P_1$ is chosen as TTP by default when both garblers identify $P_3$ to be corrupt. While the formal description of the protocol (Fig. 5), proof of correctness and proposed optimizations incorporated in our implementation are presented below, the security proof appears in the full version [19].

## 6.1 Correctness and security

Below we give the proof of correctness.

Lemma 6.1. *A pair of honest parties can never be in conflict.*

*Proof.* It is easy to note that a pair of honest garblers will never be in conflict since the message $\mathcal{B}$ broadcast by them in Round 2 must be identical. Next, a garbler, say $P_1$ and evaluator $P_3$ would be

**Figure 5: Protocol** g3PC

**Inputs:** Party $P_\alpha$ has $x_\alpha$ for $\alpha \in [3]$.
**Common Inputs:** Same as f3PC.
**Output:** $y = C(x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3 \oplus x_4)$
**Primitives:** A garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ that is correct, private and authentic with the property of soft decoding, a NICOM $(\mathsf{Com}, \mathsf{Open})$ and a PRG G.

**Round 1:**

- $P_1$ chooses random seed $s \in_R \{0,1\}^\kappa$ for G and sends $s$ to $P_2$.
- $P_3$ picks $x_{31}, x_{32} \in_R \{0,1\}^\ell$ with $x_3 = x_{31} \oplus x_{32}$. $P_3$ samples pp for NICOM and generates $(c_{31}, o_{31}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{31})$, $(c_{32}, o_{32}) \leftarrow \mathsf{Com}(\mathsf{pp}, x_{32})$, broadcasts $\{\mathsf{pp}, c_{31}, c_{32}\}$ and sends $(x_{31}, o_{31})$, $(x_{32}, o_{32})$ to $P_1, P_2$ respectively.

**Round 2:**

- $P_i (i \in [2])$ broadcasts $(\mathtt{Conflict}, P_3)$ if $\mathsf{Open}(c_{3i}, o_{3i}) \neq x_{3i}$. Else, it does the following:
  - Compute GC $(\mathsf{C}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, C)$ using randomness from $G(s)$. Assume $\{e^0_\alpha, e^1_\alpha\}_{\alpha\in[\ell]}$, $\{e^0_{\ell+\alpha}, e^1_{\ell+\alpha}\}_{\alpha\in[\ell]}$, $\{e^0_{2\ell+\alpha}, e^1_{2\ell+\alpha}\}_{\alpha\in[2\ell]}$ correspond to the encoding information for the input of $P_1, P_2$ and the shares of $P_3$ respectively (w.l.o.g).
  - Compute permutation strings $p_1, p_2 \in_R \{0,1\}^\ell$ for the garblers' input wires and generate commitments to $e$ using randomness from $G(s)$. For $b \in \{0,1\}$, $(c^b_\alpha, o^b_\alpha) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p_1^\alpha \oplus b}_\alpha)$, $(c^b_{\ell+\alpha}, o^b_{\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, e^{p_2^\alpha \oplus b}_{\ell+\alpha})$ for $\alpha \in [\ell]$ and $(c^b_{2\ell+\alpha}, o^b_{2\ell+\alpha}) \leftarrow \mathsf{Com}(\mathsf{pp}, e^b_{2\ell+\alpha})$ for $\alpha \in [2\ell]$. Set $\mathcal{B}_i = \left\{\mathsf{C}, \{c^b_\alpha\}_{\alpha\in[4\ell], b\in\{0,1\}}\right\}$. Broadcast $\mathcal{B}_i$.
  - $P_1$ computes $m_1 = x_1 \oplus p_1$ and sends to $P_3$: the openings of the commitments corresponding to $(x_1, x_{31})$ i.e $\{o^{m_1^\alpha}_\alpha, o^{x_{31}^\alpha}_{2\ell+\alpha}\}_{\alpha\in[\ell]}$ and $m_1$. Similarly, $P_2$ computes $m_2 = x_2 \oplus p_2$ and sends to $P_3$: the openings of the commitments corresponding to $(x_2, x_{32})$ i.e $\{o^{m_2^\alpha}_{\ell+\alpha}, o^{x_{32}^\alpha}_{3\ell+\alpha}\}_{\alpha\in[\ell]}$ and $m_2$.
- Every party sets TTP as follows. If exactly one $P_i (i \in [2])$ broadcasts $(\mathtt{Conflict}, P_3)$ in Round 2, set $\mathsf{TTP} = P_{[2]\setminus i}$. If both raise conflict, set $\mathsf{TTP} = P_1$. If $\mathcal{B}_1 \neq \mathcal{B}_2$, set $\mathsf{TTP} = P_3$.

**Round 3:** If $\mathsf{TTP} = \emptyset$, $P_3$ does the following:

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

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

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

**Round 5:** TTP computes $x_{3i} = \mathsf{Open}(c_{3i}, o_{3i})$ using openings sent by $P_1, P_2$ (if available), else uses the openings sent by $P_3$. If valid opening is not received, a default value is used for shares of $x_3$. Compute $y = f(x_1, x_2, x_{31} \oplus x_{32})$ and send $y$ to others.
Every party computes output as follows. If $y = \perp$ and received $y'$ from TTP, set $y = y'$.

in conflict only if one of the following hold: (a) The commitment and opening of the input share sent by $P_3$ to $P_1$ is inconsistent (b) $P_1$'s opening of committed encoded input of garbled circuit sent to $P_3$ fails. It is easy to check that the above cannot occur for honest $P_1, P_3$. □

Lemma 6.2. *An honest evaluator either evaluates GC successfully at the end of round 2 or a* TTP *is established latest by Round 3.*

*Proof.* Consider an honest $P_3$. If a garbler raises a conflict with $P_3$ in Round 2, then his co-garbler is established as the TTP. Else, if $P_3$ receives broadcast and pairwise messages as per the protocol in round 2, then $P_3$ evaluates the circuit. On the other hand, if $P_3$ discovers that the broadcast messages sent by the garblers do not match, then $P_3$ is unanimously established as the TTP. Finally, in case $P_3$ locally identifies one of the garblers to be corrupt due to inconsistent/invalid pairwise message received in round 2, he raises a conflict, establishing the other garbler as the TTP. Thus the lemma holds. □

Theorem 6.3. *The protocol* g3PC *is correct i.e output obtained by the parties corresponds to a valid computation performed on unique set of inputs.*

*Proof.* We analyze the cases based on whether TTP is established during the protocol or not. If not, since none of the garblers raised a conflict with $P_3$ in Round 2, each of them must have a valid opening corresponding to $P_3$'s public commitment of its input shares. In such a case, these shares constitute $P_3$'s committed input. With respect to garblers, input labels sent by them in round 2 corresponding to their own input establish their committed inputs. It now follows from correctness of garbling and authenticity (potentially corrupt $P_3$ could not have forged $\mathbf{Y}$) that the output obtained by all corresponds to the evaluation of garbled circuit on above mentioned committed

inputs. We now consider the case when TTP is established. Here, the inputs sent by garblers on clear to the TTP constitute their committed inputs. The committed input of $P_3$ depends on whether the TTP is established during or after Round 2. In the former where none of the garblers raised conflict in Round 2, it is clear from the protocol description that $P_3$'s committed input is based on its shares distributed in Round 1 (enforced by binding of commitment on input shares). Else, the committed input of $P_3$ is considered as the one sent on clear to the TTP. Finally, the correctness of output computation based on committed inputs follows from the fact that the TTP must be honest (Lemma 6.1 shows that the pair of parties in conflict must involve the corrupt). □

While the full proof of security appears in the full version [19], the intuition on why the protocol achieves guaranteed output delivery and the theorem statement follow. Based on whether the evaluator is honest or corrupt, guaranteed output delivery is argued below. By Lemma 6.2, an honest evaluator either identifies a TTP or evaluates the GC successfully at the end of round 2. If evaluation is performed, then an honest evaluator would obtain output by soft decoding and enable the garblers to get output by sending the encoded output. If TTP is identified by an honest evaluator all parties accept the output sent by the TTP. Next, consider a corrupt evaluator. In case a corrupt evaluator does not communicate the encoded output to the garblers or sends an invalid $\mathbf{Y}$, then the garblers would unanimously identify the evaluator to be corrupt. Then, $P_1$ would be chosen as a TTP and eventually each party receives the output through the computation performed by TTP. Even in the case when a corrupt evaluator falsely raises a conflict, the TTP chosen by him must be honest and each party would obtain the output from the TTP.

Theorem 6.4. *Assuming one-way functions, protocol* g3PC *securely realizes the functionality* $\mathcal{F}_{\mathsf{GOD}}$ *(Fig. 9) against a malicious adversary that corrupts at most one party.*

## 6.2 Optimizations

We propose several optimizations for g3PC to reduce its communication. Firstly, since broadcast communication is considered more expensive than private communication, a broadcast of a message, say $m$ is replaced with broadcast of $\mathsf{H}'(m)$, where $\mathsf{H}'$ denotes a collision-resistant hash while the message $m$ is sent privately over point-to-point channel to the receiver. Besides, the trick described for fair3PC (Section 3.2) can be applied where the common message of garblers $\mathcal{B}$ is divided into equal halves $\mathcal{B} = \mathcal{B}^1||\mathcal{B}^2$; each garbler sends one part on clear and the other in compressed form. Second, we elaborate on the optimization applied to broadcast of $\mathbf{Y}$ in round 3 by $P_3$: $P_3$ broadcasts $\mathsf{H}'(\mathbf{Y})$ where $\mathbf{Y}$ denotes the encoded output comprising of concatenation of the output label of each output wire obtained by GC evaluation. Additionally, $P_3$ sends $\mathbf{Y}$ privately to each of the garblers enabling them to compute the hash of the message received privately and check against the broadcast message to conclude its consistency. Thus, the optimization applied on broadcast of $\mathcal{B}$ and $\mathbf{Y}$ makes broadcast independent of circuit size. Finally, we point that the description of protocol in Figure 5 includes certain redundancies such as a party established as TTP sending message to itself and the protocol proceeding till the last round even in cases where termination can occur earlier. This was done only to keep the protocol description simple and facilitate better understanding. In the implementation, the redundant messages are avoided. Further, when TTP is established in round 2 itself, round 3 can be skipped and the last two rounds executed, enabling the protocol to terminate within 4 rounds.

# 7 EXPERIMENTAL RESULTS

In this section, we provide empirical results for our protocols. We use the circuits of AES-128, SHA-256 and MD5 as benchmarks. We start with the description of the setup environment, both software and hardware.

*Hardware Details.* We have experimented both in LAN and WAN setting. The specifications of our systems used for LAN include 32GB RAM; an Intel Core i7-7700-4690 octa-core CPU with 3.6 GHz processing speed. The hardware supports AES-NI instructions. For WAN setting, we use Microsoft Azure Cloud Services with machines located in West USA, East Asia and India. Our 3PC protocols have exactly one party at each location while for 4PC results, two of the four parties are located in East Asia and one party each in West USA and India. We used machines with 1.75GB RAM and single core processor. The bandwidth is limited to 100Mbps for

**Table 1: Computation time (CT), Runtime for LAN (LAN), WAN (WAN) and Communication (CC) for the 3PC of [51]**

| Circuit | CT( ms) | | LAN( ms) | | WAN( s) | | CC( KB) | |
|---|---|---|---|---|---|---|---|---|
| | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ ( s) | $P_1/P_2$ | $P_3$ |
| AES | 0.96 | 0.72 | 1.19 | 0.86 | 0.62 | 1.04 | 153.2 | 2.1 |
| SHA-256 | 11.36 | 9.4 | 13.3 | 10.7 | 1.05 | 1.65 | 3073.6 | 4.5 |
| MD5 | 4.5 | 3.0 | 4.9 | 3.9 | 0.83 | 1.24 | 1036.4 | 2.5 |

**Table 2: Computation time (CT), Runtime for LAN (LAN), WAN (WAN) and Communication (CC) for** f3PC **protocol**

| Circuit | CT( ms) | | LAN ( ms) | | WAN ( s) | | CC( KB) | |
|---|---|---|---|---|---|---|---|---|
| | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ |
| AES | 1.04 | 0.74 | 1.17 | 1.0 | 0.83 | 1.27 | 161.55 | 2.27 |
| SHA-256 | 11.55 | 9.5 | 13.6 | 12.5 | 1.65 | 1.97 | 3089.7 | 4.5 |
| MD5 | 4.61 | 3.05 | 4.96 | 4.32 | 1.39 | 1.54 | 1044.93 | 2.52 |

**Table 3: Computation time (CT), Runtime for LAN (LAN), WAN (WAN) and Communication (CC) for** g4PC **protocol**

| Circuit | CT( ms) | | | LAN( ms) | | | WAN( s) | | | CC( KB) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $P_1/P_2$ | $P_3$ | $P_4$ | $P_1/P_2$ | $P_3$ | $P_4$ | $P_1/P_2$ | $P_3$ | $P_4$ | $P_1/P_2$ | $P_3$ | $P_4$ |
| AES | 0.95 | 0.8 | 0.04 | 1.21 | 0.96 | 0.27 | 0.78 | 1.08 | 0.47 | 163.3 | 8.1 | 2.1 |
| SHA-256 | 11.3 | 9.72 | 0.09 | 13.67 | 12.06 | 0.54 | 1.86 | 2.0 | 0.54 | 3091.9 | 14.1 | 2.1 |
| MD5 | 4.42 | 3.03 | 0.07 | 5.05 | 4.1 | 0.43 | 1.24 | 1.66 | 0.52 | 1046.8 | 8.13 | 2.1 |

**Table 4: Computation time (CT), Runtime for LAN (LAN) and Communication (CC) both over private (pp) and broadcast (bc) channels for** g3PC **protocol**

| Circuit | CT( ms) | | LAN( ms) | | pp CC( KB) | | bc CC( KB) | |
|---|---|---|---|---|---|---|---|---|
| | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ | $P_1/P_2$ | $P_3$ |
| AES | 1.12 | 0.9 | 2.62 | 2.58 | 153.36 | 2.23 | 0.032 | 0.06 |
| SHA-256 | 11.63 | 9.76 | 16.25 | 13.8 | 3074.16 | 4.62 | 0.032 | 0.06 |
| MD5 | 4.73 | 3.22 | 7.18 | 5.88 | 1036.66 | 2.51 | 0.032 | 0.06 |

**Table 5: The average computation time (aCT), runtime in LAN (aLAN), WAN (aWAN) and communication (aCC) per party for [51] and our protocols. The figures in bracket indicate the increase for the worst case 5-round runs of g4PC and g3PC.**

| Circuit | aCT( ms) | | | | aLAN( ms) | | | | aWAN( s) | | | aCC( KB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | [51] | f3PC | g4PC | g3PC | [51] | f3PC | g4PC | g3PC | [51] | f3PC | g4PC | [51] | f3PC | g4PC | g3PC |
| AES | 0.88 | 0.94 | 0.69 | 1.04 | 1.08 | 1.11 | 0.91 | 2.60 | 0.76 | 0.97 | 0.78 (+.49) | 102.83 | 108.46 | 84.2 (+.01) | 103.02 (+.02) |
| SHA-256 | 10.70 | 10.87 | 8.1 | 11.01 | 12.43 | 13.23 | 9.98 | 15.43 | 1.25 | 1.75 | 1.56 (+.52) | 2050.56 | 2061.3 | 1550 (+.1) | 2051.02 (+.08) |
| MD5 | 4.0 | 4.09 | 2.98 | 4.22 | 4.56 | 4.74 | 3.65 | 6.74 | 0.96 | 1.44 | 1.16 (+.49) | 691.76 | 697.46 | 525.97 (+.03) | 691.98 (+.09) |

the WAN network between the machines in West USA and East Asia and it is limited to 8Mbps from the machine in India. Before running our experiments, we measured sample round trip delay between India-West USA, India-East Asia and East Asia-West USA for communication of one byte of data. These values average to 0.42 s, 0.14 s and 0.18 s respectively.

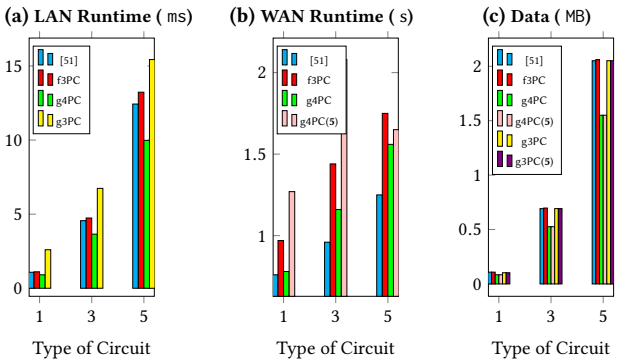
*Software Details.* For efficient implementation, the garbling technique used throughout is that of Half Gates [58]. The code is built on libgarble library whose starting point is the JustGarble library, both licensed under GNU GPL License. The libgarble library operates with AES-NI support from hardware. The operating system used for LAN and WAN results are Ubuntu 17.10 (64-bit) and Ubuntu 16.04 respectively. Our code follows the standards of C++11. We make use of openSSL 1.0.2g library for commitments. We use SHA-256 to implement a commitment scheme. We have benchmarked our results with 3 circuits AES, SHA-256, MD5. The circuit description is obtained as a simple text file (.txt) for implementation purposes. Communication is done with the help of sockets. We instantiate multiple threads to facilitate communication between the garblers and evaluator. The garblers also share a connection between each other to share the randomness. All our results indicate the average values over a set of 20 runs of the experiments.

*Comparison.* We compare our results with the related ones for the high-latency networks (such as the Internet) in the honest majority setting. The most relevant is that of [51] and we elaborate on the comparison with it below. With regard to the 4-party protocol of [40], it is expected to lag in performance compared to g4PC since its computation and communication is significantly higher. As per our calculations, the overhead of transmitting 12 GCs instead of 1 is more than the efficiency gain of having 2 rounds instead of 5, even with bandwidth of 100Mbps for our benchmark circuits of SHA-256 and MD5. In case of limited bandwidth of around 8Mbps, our protocol would perform better than that of [40] for all our benchmark circuits including AES. The difference in performance will be even more significant for larger circuits or when multiple MPC executions are run in parallel. Another work close to our setting is that of [21] that explores 5PC in the honest majority setting. Similar to [51], it only provides selective abort. It uses distributed garbling and requires 8 rounds. Our 3 party and 4 party protocols perform better than the protocol of [21], in spite of achieving better security notions of fairness and guaranteed output delivery. The total communication for any of our protocol constitutes only 1 - 3.5 % of the total communication of their implementation in the malicious setting and 3 - 6 % of the total communication of their implementation in the semi-honest setting.

For comparing with [51], four parameters are considered– computation time (**CT**), communication cost (**CC**) and runtime both in LAN (**LAN**) and WAN (**WAN**). The LAN and WAN runtime are computed by adding the computation time and the corresponding network time. Noting that the roles of the parties in the protocols are asymmetric, we show the computation time, LAN and WAN runtime and communication cost separately for the parties with distinct roles. The trend of WAN runtime across the tables indicates the influence of round complexity and the location of servers. For a fair comparison with our protocols, we instantiate the protocol of [51] in our environment and the results appear in Table 1. The results for our 3PC with fairness, 4PC (5 rounds) and 3PC with guaranteed output delivery appear in Table 2, Table 3 and Table 4 respectively. With respect to our 4-round 4PC with god, in the worst case run, we save a round at the expense of one garbled circuit over our 5-round 4PC which amounts to 72 KB – 1530 KB for the benchmark circuits. For the 3PC with guaranteed output delivery, we provide implementation result only for LAN setting where the broadcast channel is emulated using an UDP physical broadcast. We calculate separately the cost of communication over private channels and broadcast channel and demonstrate that the latter communication is independent of the circuit size. Our protocols providing guaranteed output delivery run in 3 rounds when the adversary does *not* strike. The round complexity stretches to 5 in the worst case for our 5-round protocols. Tables 3-4 show performance for the 3-round runs. With minimal communication and computation in the last two rounds, the overhead shows up mainly in the WAN runtime by a factor of half a second and communication by less than 1 KB.

For a unified comparison with [51], we compute the average of the above parameters per party for all the protocols and the results appear in Table 5. In terms of average computation time, LAN runtime and communication cost our 4PC turns out to be the winner inspite of providing the strongest notion of security. The improvement per party comes from the fact that the costs of this protocol are almost similar to the 3PC protocols inspite of having one extra party in the system. It closely trails [51] in terms of WAN runtime due to the additional communication involved in the InputCommit routine and the delayed opening of the committed decoding information both of which are not present in the protocol of [51]. Our 3PC with fairness is almost on par with [51] and yet achieves a stronger security notion. The extra overhead over [51] occurs primarily as a consequence of commitments to the decoding information and the postponed opening of decoding information by the garblers in order to achieve fairness. However, in [51], the use of soft-decoding avoided the need for additional communication to deliver the decoding information. The variation in the communication

**Figure 6: Performance Comparison (avg/party) of various Protocols for fairness and guaranteed output delivery. (5) denotes worst case execution of the protocol in consideration**




**The x-axes indicate the type of the circuit used for evaluation 1-AES, 3-MD5, 5-SHA-256. The y-axis indicates Runtime in ms, s for graphs (a), (b) respectively and communication in MB for (c).**

overhead over the circuits reflects the fact that the output size and thus the size of information (openings of the commitments) related to decoding information are different over the circuits. For example, the SHA-256 has 256 bit output, whereas the output size of AES is half of it. Therefore, the communication overhead for SHA-256 for our protocol is almost double that of AES, namely 10.74 KB vs. 5.63 KB. The WAN runtime overhead reflects the increased round requirement of our fair protocol. The communication overhead of our 3PC with guaranteed output delivery is almost nominal over [51] as both protocols use soft-decoding. In Table 5, we show in bracket the increase for the 5-round runs of our 4PC (5-round) and 3PC protocols providing guaranteed output delivery. The performance of our protocols compared to that of [51] is plotted in Fig. 6.

## REFERENCES


[1] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. 2014. Non-Interactive Secure Computation Based on Cut-and-Choose. In *EUROCRYPT*.

[2] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. 2017. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. In *IEEE Symposium on Security and Privacy, SP*. 843–862.

[3] Toshinori Araki, Assaf Barak, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. DEMO: High-Throughput Secure Three-Party Computation of Kerberos Ticket Generation. In *ACM CCS*. 1841–1843.

[4] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *ACM CCS*.

[5] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE. In *EUROCRYPT*.

[6] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*.

[7] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The Round Complexity of Secure Protocols (Extended Abstract). In *STOC*.

[8] Zuzana Beerliová-Trubíniová and Martin Hirt. 2007. Simple and Efficient Perfectly-Secure Asynchronous MPC. In *ASIACRYPT*.

[9] Zuzana Beerliová-Trubíniová and Martin Hirt. 2008. Perfectly-Secure MPC with Linear Communication Complexity. In *TCC*.

[10] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *CCS*.

[11] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. 2016. Optimizing Semi-Honest Secure Multiparty Computation for the Internet. *IACR Cryptology ePrint Archive* 2016 (2016), 1066.

[12] Aner Ben-Efraim and Eran Omri. 2017. Concrete efficiency improvements for multiparty garbling with an honest majority. In *LATINCRYPT*.

[13] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*.

[14] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. 2012. Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority. In *CRYPTO*.

[15] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. 2011. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT*.

[16] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Computer Security- ESORICS*.

[17] Dan Bogdanov, Riivo Talviste, and Jan Willemson. 2012. Deploying Secure Multi-Party Computation for Financial Data Analysis - (Short Paper). In *FC*.

[18] 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. 2009. Secure Multiparty Computation Goes Live. In *FC*.

[19] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. 2018. Fast Secure Computation for Small Population over the Internet. Cryptology ePrint Archive, Report 2018/710. (2018). https://eprint.iacr.org/2018/710.

[20] Ran Canetti. 2000. Security and Composition of Multiparty Cryptographic Protocols. *J. Cryptology* 13, 1 (2000).

[21] Nishanth Chandran, Juan A. Garay, Payman Mohassel, and Satyanarayana Vusirikala. 2017. Efficient, Constant-Round and Actively Secure MPC: Beyond the Three-Party Case. In *ACM CCS*.

[22] David Chaum, Ivan Damgård, and Jeroen Graaf. 1987. Multiparty Computations Ensuring Privacy of Each Party's Input and Correctness of the Result. In *CRYPTO*.

[23] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. 2014. Efficient Three-Party Computation from Cut-and-Choose. In *CRYPTO*.

[24] Richard Cleve. 1986. Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract). In *STOC*.

[25] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. 2016. Characterization of Secure Multiparty Computation Without Broadcast. In *TCC*.

[26] Ran Cohen and Yehuda Lindell. 2014. Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation. In *ASIACRYPT*.

[27] Ronald Cramer, Ivan Damgård, and Yuval Ishai. 2005. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *Theory of Cryptography*, Joe Kilian (Ed.). Springer Berlin Heidelberg.

[28] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. 1998. Non-Interactive and Non-Malleable Commitment. In *STOC*.

[29] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO*.

[30] Ivan Damgård and Claudio Orlandi. 2010. Multiparty Computation for Dishonest Majority: From Passive to Active Security at Low Cost. In *CRYPTO*.

[31] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO*.

[32] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. 2017. High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority. In *EUROCRYPT*.

[33] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. 2014. Two-Round Secure MPC from Indistinguishability Obfuscation. In *TCC*.

[34] Martin Geisler. 2007. Viff: Virtual ideal functionality framework. (2007).

[35] Oded Goldreich. 2001. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press.

[36] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*.

[37] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. 2015. Constant-Round MPC with Fairness and Guarantee of Output Delivery. In *CRYPTO*.

[38] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. 2018. Secure Computation with Low Communication from Cross-checking. *IACR Cryptology ePrint Archive* 2018 (2018), 216.

[39] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. 2015. Fast Garbling of Circuits Under Standard Assumptions. In *ACM CCS*.

[40] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. 2015. Secure Computation with Minimal Interaction, Revisited. In *CRYPTO*.

[41] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. 2008. Founding Cryptography on Oblivious Transfer - Efficiently. In *CRYPTO*.

[42] Mitsuru Ito, Akira Saito, and Takao Nishizeki. 1989. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* (1989).

[43] Marek Jawurek, Florian Kerschbaum, and Claudio Orlandi. 2013. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In *CCS*.

[44] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. 2014. Application-Scale Secure Multiparty Computation. In *ESOP*.

[45] Yehuda Lindell. 2013. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries. In *CRYPTO*.

[46] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. 277–346.
[47] Yehuda Lindell and Benny Pinkas. 2007. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT*.
[48] Yehuda Lindell and Benny Pinkas. 2012. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. *J. Cryptology* 25, 4 (2012), 680–722.
[49] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. 2015. Efficient Constant Round Multi-party Computation Combining BMR and SPDZ. In *CRYPTO*.
[50] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. 2017. EPIC: Efficient Private Image Classification (or: Learning from the Masters). Cryptology ePrint Archive, Report 2017/1190. (2017). https://eprint.iacr.org/2017/1190.
[51] Payman Mohassel, Mike Rosulek, and Ye Zhang. 2015. Fast and Secure Three-party Computation: The Garbled Circuit Approach. In *ACM CCS*.
[52] Moni Naor. 1991. Bit Commitment Using Pseudorandomness. *J. Cryptology* 4, 2 (1991).
[53] Jesper Buus Nielsen and Claudio Orlandi. 2016. Cross and Clean: Amortized Garbled Circuits with Constant Overhead. In *TCC*.
[54] Arpita Patra and Divya Ravi. 2018. On the Exact Round Complexity of Secure Three-Party Computation. *IACR Cryptology ePrint Archive* 2018 (2018), 481.
[55] Tal Rabin and Michael Ben-Or. 1989. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *STOC*.
[56] Phillip Rogaway and Thomas Shrimpton. 2004. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In *FSE*.
[57] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *FOCS*.
[58] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT*.

# A THE SECURITY MODEL

We prove the security of our protocols based on the standard real/ideal world paradigm. Essentially, security of a protocol is analyzed by comparing an adversary's behaviour in a real execution to that of an ideal execution that is considered secure by definition (in the presence of an incorruptible trusted party). In an ideal execution, each party sends its input to the trusted party over a perfectly secure channel, the trusted party computes the function based on these inputs and sends to each party its respective output. Informally, a protocol is secure if whatever an adversary can do in the real protocol (where no trusted party exists) can be done in the above described ideal computation. We refer to [20, 26, 35, 46] for further details regarding the security model.

The "ideal" world execution involves a set of parties $\mathcal{P}$ with $|\mathcal{P}| = 3$ or 4 (corresponding to 3PC / 4PC), an ideal adversary $\mathcal{S}$ who may corrupt one of the parties, and a functionality $\mathcal{F}$. The "real" world execution involves the PPT set of parties $\mathcal{P}$, and a real world adversary $\mathcal{A}$ who may corrupt one of the parties. We let $\text{IDEAL}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)$ denote the output pair of the honest parties and the ideal-world adversary $\mathcal{S}$ from the ideal execution with respect to the security parameter $1^\kappa$ and auxiliary input $z$. Similarly, let $\text{REAL}_{\Pi,\mathcal{A}}(1^\kappa, z)$ denote the output pair of the honest parties and the adversary $\mathcal{A}$ from the real execution with respect to the security parameter $1^\kappa$ and auxiliary input $z$.

*Definition A.1.* For $n \in \mathbb{N}$, let $\mathcal{F}$ be a functionality and let $\Pi$ be a 3/4-party protocol. We say that $\Pi$ *securely realizes* $\mathcal{F}$ if for every PPT real world adversary $\mathcal{A}$, there exists a PPT ideal world adversary $\mathcal{S}$, corrupting the same parties, such that the following two distributions are computationally indistinguishable: $\text{IDEAL}_{\mathcal{F},\mathcal{S}} \overset{c}{\approx} \text{REAL}_{\Pi,\mathcal{A}}$.

**Target Functionalities.** Taking motivation from [26, 37], we define two ideal functionalities $\mathcal{F}_{\mathsf{Fair}}, \mathcal{F}_{\mathsf{GOD}}$ in Figures 8, 9 for secure 3PC of a function $f$ with fairness and guaranteed output delivery respectively. The functionalities can be defined similarly for 4PC.

**Figure 8: Ideal Functionality $\mathcal{F}_{\mathsf{Fair}}$**

Each honest party $P_i$ ($i \in [3]$) sends its input $x_i$ to the functionality. Corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary. When sending the inputs to the trusted party, the adversary is allowed to send a special `abort` command as well.

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

**Output:** If there exists $i \in [3]$ such that $x_i' = \mathtt{abort}$, send $(\mathtt{Output}, \perp)$ to all the parties. Else, send $(\mathtt{Output}, y)$ to party $P_i$ for every $i \in [3]$, where $y = f(x_1', x_2', x_3')$.

**Figure 9: Ideal Functionality $\mathcal{F}_{\mathsf{GOD}}$**

Each honest party $P_i$ sends its input $x_i$ to the functionality. Corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary.

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

**Output:** Compute $y = f(x_1', x_2', x_3')$ and send $(\mathtt{Output}, y)$ to party $P_i$ for every $i \in [3]$.

# B PRIMITIVES

## B.1 Properties of Garbling Scheme

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

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

*Definition B.2.* (Privacy) A garbling scheme $\mathcal{G}$ is private if for all input lengths $n \le \text{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$, there exists a PPT simulator $\mathcal{S}_{\mathbf{priv}}$ such that for all inputs $x \in \{0,1\}^n$, for all probabilistic polynomial-time adversaries $\mathcal{A}$, the following two distributions are computationally indistinguishable:

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

*Definition B.3.* (Authenticity) A garbling scheme $\mathcal{G}$ is authentic if for all input lengths $n \le \text{poly}(\kappa)$, circuits $C : \{0,1\}^n \to \{0,1\}^m$,

inputs $x \in \{0,1\}^n$, and all PPT adversaries $\mathcal{A}$, the following probability is negligible in $\kappa$:

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

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

*Properties.*

- *Correctness:* For all pp, $x \in \mathcal{M}$ and $r \in \mathcal{R}$, if $(c,o) \leftarrow \mathsf{Com}(x;r)$ then $\mathsf{Open}(c,o) = x$.
- *Binding:* For all PPT adversaries $\mathcal{A}$, it is with negligible probability (over uniform choice of pp and the random coins of $\mathcal{A}$) that $\mathcal{A}(\mathsf{pp})$ outputs $(c,o,o')$ such that $\mathsf{Open}(c,o) \neq \mathsf{Open}(c,o')$ and $\perp \notin \{\mathsf{Open}(c,o), \mathsf{Open}(c,o')\}$
- *Hiding:* For all PPT adversaries $\mathcal{A}$, all pp, and all $x, x' \in \mathcal{M}$, the following difference is negligible: $\left|\Pr_{(c,o)\leftarrow\mathsf{Com}(x)}[\mathcal{A}(c)=1] - \Pr_{(c,o)\leftarrow\mathsf{Com}(x')}[\mathcal{A}(c)=1]\right|$

We use a NICOM with the above properties for our 3-party protocols. The NICOM (sCom, sOpen) with strong binding is used in our 4-party protocols. It has the same properties except that binding is defined over all pp (not just uniform choice of pp).

*Instantiations.* Here we present two instantiations of NICOM borrowed from [51]. In the random oracle model, the commitment is defined as $(c,o) = (\mathsf{H}(x||r), x||r) = \mathsf{Com}(x;r)$. The pp can in fact be empty. We use this commitment scheme for implementation purposes where the random oracle is realized via SHA-256.

In the standard model, we can use a multi-bit variant of Naor's commitment [52]. For $n$-bit messages, we need a $\mathsf{pp} \in_R \{0,1\}^{4n}$. Let $\mathrm{G} : \{0,1\}^n \rightarrow \{0,1\}^{4n}$ be a pseudorandom generator, and let $\mathsf{Pad} : \{0,1\}^n \rightarrow \{0,1\}^{4n}$ be the function that prepends $3n$ zeroes to its argument. Then the commitment scheme is:

- $\mathsf{Com}(x;r)$: set $\mathrm{C} = \mathrm{G}(r) + \mathsf{pp} \cdot \mathsf{Pad}(x)$, with arithmetic in $\mathbb{GF}(2^{4n})$; set $o = (r,x)$.
- $\mathsf{Open}(c, o = (r,x))$: return $x$ if $c = \mathrm{G}(r) + \mathsf{pp} \cdot \mathsf{Pad}(x)$; otherwise return $\perp$.

Note that binding of Naor-based instantiation holds over uniform choice of pp. However, the random-oracle based instantiation satisfies the stronger binding property needed in our 4-party protocol. We now present an instantiation of NICOM (sCom, sOpen) based on injective one-way function (alternately one-way permutation) where binding holds even for adversarially chosen pp. In the standard model, we can use the following bit-commitment scheme from any injective one-way function. Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be a one-way permutation and $h : \{0,1\}^n \rightarrow \{0,1\}$ a hard core predicate for $f(\cdot)$. Then the commitment scheme for a single bit $x$ is:

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

## B.3 Equivocal Non-interactive Commitment (eNICOM)

*Properties.*

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

*Instantiations.* The folklore commitment scheme $(c,o) = (\mathsf{H}(x||r), x||r) = \mathsf{Com}(x;r)$ in the random oracle model supports equivocation via programmability of the random oracle. The $(\mathsf{epp}, t = (t_1, t_2))$ can in fact be empty. For empirical purposes alone, we rely on this random oracle based commitment scheme where the random oracle is realized using SHA-256.

In the standard model, we present the equivocal bit commitment scheme of [28], which is based on Naor's commitment scheme [52] for single bit message. This scheme avoids the use of public-key primitives. Let $\mathrm{G} : \{0,1\}^n \rightarrow \{0,1\}^{4n}$ be a pseudorandom generator.

- $\mathsf{eGen}(1^\kappa)$: set $(\mathsf{epp}, t_1, t_2) = ((\sigma, \mathrm{G}(r_0), \mathrm{G}(r_1)), r_0, r_1)$, where $\sigma = \mathrm{G}(r_0) \oplus \mathrm{G}(r_1)$
- $\mathsf{eCom}(x;r)$: set $c = \mathrm{G}(r)$ if $x = 0$, else $c = \mathrm{G}(r) \oplus \sigma$; set $o = (r,x)$
- $\mathsf{eOpen}(c, o = (r,x))$: return $x$ if $c = \mathrm{G}(r) \oplus x \cdot \sigma$ (where $(\cdot)$ denotes multiplication by constant); else return $\perp$.
- $\mathsf{Equiv}(c = \mathrm{G}(r_0), \perp, x, (t_1, t_2))$: return $o = (r,x)$ where $r = t_1$ if $x = 0$, else $r = t_2$. Both $t_1, t_2$ are needed to perform equivocation.