

Round Optimal MPC With Provable Cheater Identification

Yashvanth Kondi¹, Divya Ravi²,
Jure Sternad³, and Sophia Yakoubov³

¹ Silence Laboratories, Singapore; yash@ykondi.net

² University of Amsterdam, Netherlands; d.ravi@uva.nl

³ Aarhus University, Denmark; {jsternad, sophia.yakoubov}@cs.au.dk

Abstract. Secure multiparty computation with provable identifiable selective abort (PISA), introduced by Kondi and Ravi (CCS 2025), guarantees that every honest party either obtains the computation output or a certificate that convinces any external auditor that a specific party cheated. Crucially, honest parties need not agree: some may get output while others receive evidence of cheating. In contrast, secure computation with identifiable abort (IA) requires just such unanimity; however, in the event that an honest party doesn't get output, it need merely be able to point to a cheater — not to prove their guilt.

Secure computation with IA classically requires broadcast. Kondi and Ravi demonstrated that secure computation with PISA, on the other hand, can be built over point-to-point channels alone.

While PISA is feasible exactly when guaranteed output delivery is, i.e. for $t < n/2$, the best known construction over point-to-point channels requires six rounds. We settle the round complexity of PISA for all but a narrow band of thresholds. We prove that two rounds are impossible for $t \geq n/3$, even with arbitrary setup and computational assumptions; we give a three-round protocol for any $t < n/2$, which is therefore round-optimal for $n/3 \leq t < n/2$; and we give a two-round protocol for $t < n/4$. The two-round protocol introduces a new primitive called one-or-nothing secret sharing with omissions.

Table of Contents

ROUND OPTIMAL MPC WITH PROVABLE CHEATER IDENTIFICATION . . .	1
<i>Yashvanth Kondi[✉], Divya Ravi[✉], Jure Sternad[✉], and Sophia Yakoubov[✉]</i>	
1 Introduction	2
1.1 Our Results	4
1.2 Technical Overview	5
1.3 Related Work	8
2 Background: PISA and BC – PISA	8
2.1 Ideal Functionality for PISA	8
2.2 Broadcast with PISA	10
3 Preliminaries	12
3.1 Public Key Encryption	12
3.2 Secret Sharing	13
3.3 Digital Signatures	14
3.4 Garbling Scheme	15
3.5 Non-Interactive Zero-Knowledge Arguments of Knowledge	16
4 Impossibility of Two Round PISA With $n \leq 3t$	17
5 Three Round PISA With Honest Majority	20
6 Two Round PISA With Honest Super-majority	23
6.1 One-or-Nothing Secret Sharing with Omissions (1or0wo)	23
6.2 MPC Protocol	31
6.3 Security Proof	36

1 Introduction

Secure multiparty computation (MPC) [CDvdG88,GMW87,Yao86] allows a set of n mutually distrusting parties to jointly evaluate a function of their private inputs while revealing nothing beyond the output, even when up to t of them are corrupt. A *maliciously* secure protocol even guards against arbitrary protocol deviations by corrupt parties. While the baseline guarantee is that of privacy, a full system needs well-specified mechanisms to thwart adversaries that attempt to prevent its operation altogether. There is a spectrum of methods by which this is achieved in deployments, from heuristic to provable. For instance, a “secure with abort” threshold signing protocol may simply be restarted with an alternative signing quorum, with participants determined by out of band methods such as reputation scores. A distributed sampling protocol for structured reference strings may be configured to proceed via globally visible, publicly verifiable updates, so that it suffices for only one trustworthy update to be visibly included in order for the reference string to be usable. While there is a rich theory of *Guaranteed Output Delivery* (GOD) that studies this problem formally, real deployments can introduce layers of complexity.

One indirect method by which guaranteed output can be achieved is via *Identifiable Abort* (IA)—intuitively, an IA protocol identifies the offending party in the event of a protocol failure, so that this party can either be removed or punished. This principle is already embodied in how certain blockchains employ “slashing” mechanisms. However, the canonical formulation of IA in the literature entails the use of broadcast [CL17], which poses several challenges in practice. As Kondi and Ravi [KR25] surveyed, using an appropriate instantiation of broadcast can be prohibitively expensive, leaving deployments to resort to heuristics. Kondi and Ravi further went on to disentangle what role precisely broadcast serves towards achieving IA, and defined *Provable Identifiable Selective Abort* (PISA) as a notion that permits efficient instantiations over point-to-point channels alone, while retaining the utility of cheater identification. Roughly, a protocol that achieves PISA enables parties to certify (even to an external auditor) the identity of *some* corrupt party that disrupted the protocol—honest parties may not be in agreement about the identity of the corrupt party, or indeed whether the protocol even aborted. In many contexts, such as those where a slashing oracle can punish misbehaviour, the threat of punitive action can be sufficient to incentivize adherence to the protocol.

The relationship between PISA and other security notions is subtle. Of course, its lack of a consensus condition means that it can not subsume traditional IA. However, its requirement of publicly verifiable certificates means that it is not subsumed by IA either. Indeed, Kondi and Ravi proved that PISA is infeasible to achieve in the $t \geq n/2$ setting (where n is the number of participants and t is an upper bound on how many participants may be malicious). While they do provide constructions that avoid broadcast in the $t < n/2$ setting, there is not yet a tight characterization of what switching to PISA can buy in terms of efficiency. Given that the PISA formulation is in large part motivated by efficiency and implementability over point-to-point channels alone, it is important to establish to what extent and under what constraints PISA can provide relief from broadcast.

One particularly important metric in practice is *round complexity*, i.e. the number of sequential message exchanges required by a protocol. The minimal number of rounds required to achieve different security notions is extensively studied in the literature [GIKR02,MW16,GGHR14,GLS15,GS18,BL18,ACGJ18]; however, the round complexity landscape for PISA is relatively unexplored. In the $t < n/2$ setting, Kondi and Ravi gave a generic compiler that transforms any r -round IA protocol using broadcast into a $(2r + 2)$ -round PISA protocol over point-to-point channels. The compiler emulates each broadcast with a two-round *broadcast with PISA* (BC – PISA) sub-protocol, and spends two additional rounds propagating certificates. Instantiated with a round-optimal two-round IA protocol in the broadcast model [CGZ20], this yields a *six*-round PISA protocol, when both PKI and CRS are available. This leaves a significant gap between the two rounds needed with broadcast and the six rounds known without it, and raises the central question of this work:

What is the exact round complexity of MPC with provable identifiable selective abort over point-to-point channels?

Round complexity is arguably the most important efficiency measure for MPC in the setting we consider: point-to-point channels are the setting of practical interest precisely because broadcast (or its emulation via Byzantine agreement) is round-expensive, and a PISA protocol that itself used many rounds would forfeit much of the benefit. The question is also natural from a foundational perspective, as the round complexity of MPC with various output guarantees and various communication models has been characterized almost completely [GLS15,ACGJ18,PR18,CGZ20,DMR⁺21,DRSY23a], and PISA is the one natural guarantee in the broadcast-free setting whose round complexity has remained open.

1.1 Our Results

Before presenting our results, we recap (informally) the guarantees of a protocol achieving PISA (see Section 2.1 for a formal definition). *Correctness* ensures that every honest party either obtains the correct output or a proof-of-cheating certificate against some corrupt party, which can be verified by any external auditor. However, honest parties need not agree on their verdicts. *Non-frameability* ensures that a corrupt party cannot produce a valid certificate against an honest party. *Privacy* is standard, ensuring that the adversary learns nothing beyond the output.

We resolve the round complexity of PISA MPC for all corruption thresholds except a narrow band, in the same setting as [KR25]: point-to-point channels, a PKI and a CRS, and no broadcast. Our results are as follows.

1. **Two rounds are impossible for $n/3 \leq t$ (Section 4).** We show that there exist functionalities that cannot be computed with PISA security in two rounds whenever $t \geq n/3$. Concretely, we prove that no two-round three-party protocol tolerating a single corruption computes the simple “selection” function in which $\mathcal{P}_1$ chooses which of $\mathcal{P}_2$ ’s and $\mathcal{P}_3$ ’s inputs to learn; the general statement follows by a standard player-partitioning argument. This is in sharp contrast with the broadcast model, where two rounds suffice for GOD for all $t < n/2$ [GLS15], even if broadcast is available only in the first round [DMR⁺21]. Hence, for $n/3 \leq t < n/2$, giving up broadcast in exchange for publicly verifiable cheater identification costs at least one additional round. The argument is information-theoretic in flavor and holds regardless of setup and computational assumptions.
2. **Three rounds suffice for $t < n/2$ (Section 5).** We give a three-round PISA protocol for any honest majority, improving on the six rounds obtained from the compiler of [KR25]. Together with the impossibility of PISA for $t \geq n/2$ [KR25] and our two-round lower bound, this is round-optimal for the entire range $n/3 \leq t < n/2$. The protocol is obtained by revisiting the compiler of [KR25] and identifying which of its rounds are truly necessary: if

Corruption threshold	2 rounds	3 rounds	Reference
$t \geq n/2$	X	X	[KR25]
$n/3 \leq t < n/2$	X	✓	Section 4, Section 5
$n/4 \leq t < n/3$	?	✓	Section 5
$t < n/4$	✓	✓	Section 6

Table 1: Feasibility of PISA MPC over point-to-point channels with PKI and CRS, as a function of the corruption threshold. Prior to this work, the best upper bound for every feasible threshold was six rounds [KR25].

the underlying protocol uses broadcast only in its first round and cannot be made to abort by misbehavior in its last round, then both sources of overhead in the compiler disappear. The two-round GOD protocol of Gordon, Liu and Shi [GLS15], as modified in [DMR⁺21] to use broadcast only in the first round, has exactly these properties. Like [GLS15], this construction assumes a structured PKI (supporting threshold fully homomorphic encryption) together with a CRS.

3. **Two rounds suffice for $t < n/4$ (Section 6).** Finally, we show that when the adversary controls fewer than a quarter of the parties, PISA is achievable in *two* rounds, matching the round complexity of MPC in the broadcast model. We follow the garbled-circuit paradigm of [DMR⁺21, DRSY23b]: each party sends its first-round message of an underlying two-round GOD protocol together with a garbled circuit computing its second-round message, and in the second round parties “vote” on which first-round messages they received, so that at most one label per wire is ever reconstructed. In [DMR⁺21, DRSY23b] votes are cast over broadcast and a non-voter is exposed to all. Over point-to-point channels a corrupt party can withhold its vote without leaving a provable trace, and a few non-receipt claims cannot be held against a sender without enabling framing, so an honest party must reconstruct from an incomplete set of votes. Our central new tool, *One-or-Nothing Secret Sharing with Omissions (1or0wo)*, is a variant of the one-or-nothing secret sharing of [DMR⁺21] that does exactly this while still preventing a corrupt dealer from getting two secrets reconstructed. Guaranteeing both from fewer votes is what forces $n > 4t$. We also require $n = O(\log(\lambda))$, since in one of our proofs of security, the reduction must make a guess the success of which becomes negligible for larger n .

Table 1 summarizes the state of affairs. The only remaining gap is the two-round question for $n/4 \leq t < n/3$, which we leave open.

1.2 Technical Overview

Lower bound. Our impossibility result exploits the tension between non-frameability and privacy of PISA in a two-round protocol. Consider a three-party function where $\mathcal{P}_1$ holds a selection bit $b \in \{2, 3\}$ and learns the input x_b of $\mathcal{P}_b$. Suppose

corrupt $\mathcal{P}_3$ behaves honestly except that it withholds its second-round message from $\mathcal{P}_1$. Then $\mathcal{P}_1$ cannot output a certificate against $\mathcal{P}_3$: a corrupt $\mathcal{P}_1$ that merely *pretends* not to have heard from an honest $\mathcal{P}_3$ has exactly the same view, and non-frameability forbids it from producing a valid certificate. Since it cannot produce a certificate, honest $\mathcal{P}_1$ must therefore output x_b from a view that does not include $\mathcal{P}_3$'s (or, symmetrically, $\mathcal{P}_2$'s) second-round message. The crux is that first-round messages of $\mathcal{P}_2$ and $\mathcal{P}_3$ are independent of b , and each of their second-round messages depends only on what $\mathcal{P}_1$ sent to that party in the first round. A corrupt $\mathcal{P}_1$ can thus run the protocol with $b = 2$ towards $\mathcal{P}_2$ and $b = 3$ towards $\mathcal{P}_3$, and assemble two views: one that is entirely consistent with $b = 2$ and excludes only $\mathcal{P}_3$'s second-round message (which, as explained above, allows $\mathcal{P}_1$ to learn x_2) and one that is entirely consistent with $b = 3$ and excludes only $\mathcal{P}_2$'s second-round message (which allows $\mathcal{P}_1$ to learn x_3). Learning *both* inputs is a clear violation of privacy, demonstrating that no two-round protocol can realize this function with PISA. Note that with a broadcast channel in the first round, $\mathcal{P}_1$ would be unable to send inconsistent inputs to $\mathcal{P}_2$ and $\mathcal{P}_3$, which is exactly what allows two-round protocols in the broadcast model to sidestep this attack.

Three rounds for honest majority. We observe that the compiler of [KR25] that transforms any r -round IA protocol using broadcast to a $2r + 2$ round PISA MPC protocol over point-to-point channels, incurs its overhead in two places. First, every round of the underlying protocol that uses broadcast is emulated by a two-round BC – PISA sub-protocol: In this two-round sub-protocol, a dealer sends a signed message in the first round and all parties echo what they received in the second round. The guarantee is that an honest party obtains either the dealer's unique signed message or a proof-of-cheating certificate against the dealer (which can be either two conflicting signatures, or $t + 1$ signed complaints of non-receipt). Second, cheating detected in the last round, whether by the underlying IA protocol or by the BC – PISA emulation of that round, costs additional rounds: the former needs additional rounds to make the identification certifiable via $t + 1$ signed attestations, and the latter for forwarding the resulting proof-of-cheating.

We observe that both costs can be avoided by choosing the underlying protocol being compiled carefully. If broadcast is used only in the first round, only one BC – PISA emulation is required; and if the underlying protocol has *guaranteed output delivery* (instead of IA) and uses only point-to-point channels in its second round, then no honest party can be made to abort by second-round misbehavior, so any certificate ever produced originates in the first-round emulation and can be forwarded “for free” in the third round, in parallel with the second round of the underlying protocol. The protocol of [GLS15] with the modification of [DMR⁺21] satisfies exactly these requirements, and our three-round protocol follows. The security proof is a reduction to the underlying protocol that carefully filters the messages of honest parties who terminated early with a certificate.

Two rounds for honest super-majority. Without a third round there is no opportunity to forward certificates, so every honest party must reach a verdict — output or certificate — directly from the two rounds of messages it receives. Following [DMR⁺21], we compile an underlying protocol with guaranteed output delivery. In the first round, parties send the prescribed first-round messages from the underlying protocol. Since honest parties are not guaranteed to have consistent views of corrupt party messages after just one round, responding with their second-round messages from the underlying protocol is dangerous: security might not hold if honest parties respond based on inconsistent views. Instead, like [DMR⁺21], we have each party garble the circuit computing its second-round message of the underlying protocol, and distribute the input labels via a one-or-nothing secret sharing scheme: for each bit of each first-round message, every party votes (by releasing an appropriate secret key) for the value it received, and the scheme guarantees that at most one label per input wire is ever reconstructed.

In [DMR⁺21] the second round is broadcast, so a party that fails to vote is visible to all and can be identified. In our setting the second round is over point-to-point channels, and a corrupt party may simply withhold its vote from some honest parties without any provable trace. An honest party can therefore only count on $n - t$ votes, of which t may be honest votes for $\perp$ that cannot be treated as evidence against the sender (else t corrupt parties could frame an honest sender).

This leads us to define a new variant, namely, *One-or-Nothing Secret Sharing with Omissions* (**1or0wo**), which guarantees reconstruction whenever $n - 2t$ non-conflicting votes for the same value are available, all other received votes are $\perp$ and at least $n - t$ votes were received in total. The price of tolerating missing and $\perp$ votes is that a corrupt sender has more room to play with: by sending different versions of its first-round message to different honest parties, or none at all, it may try to get the labels for two different bits reconstructed. Ensuring that this fails is what forces $n > 4t$. The scheme itself is a two-level threshold sharing of each label, with shares encrypted under vote-specific public keys, just like in [DMR⁺21].

Whenever the **1or0wo** sharing scheme would not allow reconstruction of a label, the MPC protocol provides a proof-of-cheating certificate instead. If honest parties voted for conflicting values, the sender must have signed two different first-round messages, which is itself a certificate against it; if too few votes arrived, more than t signed non-receipt claims certify the sender’s non-responsiveness; and NIZK proofs ensure that every sharing was formed correctly, so that whenever the reconstruction condition is met the right label is recovered. Every honest party thus ends up either with all the labels it needs, and hence with the output, or with a certificate against a corrupt party, which is exactly the PISA guarantee.

1.3 Related Work

Cheater identification. Identifiable abort was formalized by Ishai, Ostrovsky and Zikas [IOZ14], who showed it to be achievable for any number of corruptions given broadcast. A line of work has since made IA concretely efficient [BOSS20,BMRS24,CDKs24,Riv25]; some of these additionally offer *public verifiability*, i.e., a transcript of the (broadcast) execution allows an outsider to verify which party cheated, in the spirit of publicly auditable MPC [BDO14]. All of these works rely on broadcast (or an authenticated bulletin board) to reach agreement on the cheater. PISA [KR25] differs in that it removes broadcast altogether and replaces agreement with transferable evidence; as shown in [KR25], this requires an honest majority. Our contribution is to settle the round complexity of this notion.

Round complexity of MPC. The round complexity of MPC is well understood in the broadcast model. With an honest majority, two rounds are necessary and sufficient for GOD assuming a PKI or CRS [GLS15,ACGJ18], and three rounds are needed without setup [GIKR01,PR18]. Without setup, two rounds still suffice for unanimous abort [ACGJ18], and even for identifiable abort when $t < n/3$ [KRSY26]; for $t \geq n/3$, identifiable abort is impossible in two rounds [GJPR21]. Cohen, Garay and Zikas [CGZ20] initiated the study of *broadcast-optimal* two-round MPC, characterizing which security guarantees are achievable when broadcast is available in only one of the two rounds, or in neither, in the dishonest-majority setting; Damgård et al. [DMR⁺21,DRSY23b] gave the analogous characterization for honest majority, and [DRSY23a] extended it to asynchronous point-to-point channels. In particular, [DMR⁺21] shows that when a PKI is available, two-round GOD is achievable with broadcast only in the first round, and that with no broadcast at all only *selective* abort (where only a selected subset of honest parties may obtain the output) is achievable for $t \geq n/3$ in two rounds. Our lower bound sharpens the latter for PISA: even the weak selective guarantee cannot be upgraded to a *provable* one in two rounds when $t \geq n/3$, while our two-round protocol shows that it can be for $t < n/4$. Both of our constructions build on the techniques of [DMR⁺21], in particular the first-round-broadcast variant of [GLS15] and the garbled-circuit-plus-one-or-nothing-sharing paradigm.

Roadmap. Section 2.1 recalls the definition of PISA and the BC – PISA building block from [KR25]. Standard preliminaries follow. Section 4 presents our two-round impossibility result. Section 5 presents the three-round protocol for honest majority, and Section 6 introduces the primitive One-or-Nothing Secret Sharing with Omissions and our two-round protocol for $t < n/4$.

2 Background: PISA and BC – PISA

2.1 Ideal Functionality for PISA

We recap the formal description of the ideal functionality for PISA, as in [KR25]. The parties first submit their inputs, where the corrupt parties can choose their

inputs arbitrarily. Next, the adversary first learns the output of the computation, and then determines which honest parties receive this output. Each honest party that is denied the output is instead given the identity of a corrupt party (determined by the adversary) along with the type of cheating (either non-responsiveness denoted as `nr` or active cheating denoted as `cheat`). Additionally, there is a provision to transfer this information to any third party via a *transfer* interface. The transfer interface captures auditability: it lets an honest party convince any external party, even one that did not participate in the computation, of the identity of a cheater.

Functionality $\mathcal{F}_{\text{PISA}}(n, t, f)$: Provable Identifiable Selective Abort

This functionality is parameterized by the party count n and the threshold t such that $n \geq 2t + 1$. It is accessed by parties $\{\mathcal{P}_i\}_{i \in [n]}$ at most t of whom may be corrupted by an ideal adversary $\mathcal{S}$. Let the set of honest parties be indexed by $\mathcal{H}$. It is also parameterized by a function $f : \mathbb{X}_1 \times \mathbb{X}_2 \cdots \times \mathbb{X}_n \rightarrow \mathbb{Y}$.

Computation. On receiving `(compute, sid, xi)`, where $x_i \in \mathbb{X}_i$ from every party $\mathcal{P}_i$ for $i \in [n]$,

1. Compute $y := f(\{x_i\}_{i \in [n]})$.
2. Send `(function-output, sid, y)` to $\mathcal{S}$.
3. Receive `out` = $\{\text{out}_i \in \{y, (\text{cheat}, c_i), (\text{nr}, c_i)\}\}_{i \in [n]}$ from $\mathcal{S}$ in response, where each $c_i (i \in [n])$ corresponds to an index of a corrupt party.
4. Send `(output, sid, outi)` to each $\mathcal{P}_i$. Additionally, for each `pid` corresponding to every honest party $h \in \mathcal{H}$, set and store `outpid` = `(sid, outh)`.

Transfer. Upon receiving `(transfer, sid, pid, pid*)` from both $\mathcal{P}_{\text{pid}}$ and $\mathcal{P}_{\text{pid}^*}$

1. If the party $\mathcal{P}_{\text{pid}}$ is honest and has previously been given `outpid`, set `outpid*` = `outpid` and send it to $\mathcal{P}_{\text{pid}^*}$.
2. Otherwise, receive `outpid` $\in \{y, (\text{cheat}, c), (\text{nr}, c)\}$ from $\mathcal{S}$, where c corresponds to an index of a corrupt party $\mathcal{P}_c \in \{\mathcal{P}_i\}_{i \in [n]}$, set `outpid*` = `outpid` and send it to $\mathcal{P}_{\text{pid}^*}$.

We note that the functionality as written allows cheaters as well as *computation outputs* to be certified to external parties. The latter does not harm feasibility of achieving this notion — the functionality itself could compute signed outputs for example — but may be dropped if desired for efficiency reasons.

We make use of several properties of PISA protocols, which are implied by the ideal functionality:

Non-frameability: A corrupt party should be unable to create a valid proof of cheating against an honest party.

Correctness: Each honest party outputs either the correct function output or a valid proof of cheating.

Privacy: The adversary is unable to learn anything beyond the function output.

2.2 Broadcast with PISA

Our PISA constructions rely on the building block of *Broadcast with PISA*, which was first introduced by [KR25]. Informally, *Broadcast with PISA* guarantees that every honest party either obtains a valid message signed by the dealer or outputs a proof-of-cheating certificate against the dealer (for a corrupt dealer). Some honest parties may output a certificate while others obtain the dealer's signed message. Nevertheless, consistency is preserved: no two honest parties output distinct signed messages.

Below, we recap the ideal functionality for broadcast with PISA (BC – PISA) as well as a 2-round protocol $\Pi_{\text{BC-PISA}}$ that uses *only point-to-point* channels and realizes such a functionality (almost verbatim from [KR25]).

Functionality $\mathcal{F}_{\text{BC-PISA}}(n, t, D) : \text{BC} - \text{PISA}$

This functionality is parameterized by the party count n , the threshold t such that $n \geq 2t + 1$, and the index of the dealer D . This functionality is accessed by parties $\{\mathcal{P}_i\}_{i \in [n]}$ at most t of whom may be corrupt. Let the set of honest parties be indexed by $\mathcal{H}$.

Any party, indexed by a global identifier pid , may register at any point with this functionality. Each such party $\mathcal{P}_{\text{pid}}$ is provided with an output out_{pid} which can be transferred to any other party. The set of such parties is initially only $\{\mathcal{P}_i\}_{i \in [n]}$, and can be expanded as necessary subsequently.

Deal. On receiving $(\text{deal}, \text{sid}, \text{msg})$ from $\mathcal{P}_D$ such that sid is fresh, send $(\text{deal-req}, \text{sid}, D)$ to $\mathcal{S}$. On receiving $(\text{ready}, \text{sid})$ from all honest parties,

1. If $\mathcal{P}_D$ is not corrupt, then set $\text{out} = \{\text{out}_i = \text{msg}\}_{i \in [n]}$.
2. Otherwise, receive $\text{out} = \{\text{out}_i \in \{\text{msg}, \text{cheat}, \text{nr}\}\}_{i \in [n]}$ from $\mathcal{S}$.

Send $(\text{dealt}, \text{sid}, \text{out}_i)$ to each $\mathcal{P}_i$. Additionally, for each pid corresponding to every party $h \in \mathcal{H}$, set and store $\text{out}_{\text{pid}} = (\text{sid}, \text{out}_i)$.

Transfer. Upon receiving $(\text{transfer}, \text{sid}, \text{pid}, \text{pid}^*)$ from both $\mathcal{P}_{\text{pid}}$ and $\mathcal{P}_{\text{pid}^*}$

1. If $\mathcal{P}_D$ is honest, send out_{pid} to $\mathcal{P}_{\text{pid}^*}$.
2. Otherwise, receive $\text{out}_{\text{pid}} \in \{\text{msg}^*, \text{cheat}, \text{nr}\}$ from $\mathcal{S}$ and send to $\mathcal{P}_{\text{pid}^*}$.

At a high level, the two-round protocol $\Pi_{\text{BC-PISA}}$ works as follows. In the first round, the dealer sends a signed message to all parties; in the second round, parties echo their received messages. An honest party identifies the dealer as

a cheater if it either (a) obtains two distinct messages validly signed by the dealer or (b) more than t parties report having not heard from the dealer. These cases yield proofs of active cheating **cheat** and non-responsiveness **nr**, respectively. Otherwise, honest parties obtain the dealer's unique signed message. Consistency follows from the echo phase as it ensures that no two honest parties output distinct messages.

Protocol $\Pi_{\text{BC-PISA}}$

Let n denote the number of parties, t denote the threshold such that $n \geq 2t + 1$, and the index of the dealer as D . This protocol is run by parties $\{\mathcal{P}_i\}_{i \in [n]}$. The protocol requires a PKI, where the key pair of $\mathcal{P}_i$ is given by $(\text{sk}_i^{\text{PKI}}, \text{pk}_i^{\text{PKI}})$.

In the event that any $\mathcal{P}_i$ aborts (with an accompanying certificate), it is taken by assumption that $\mathcal{P}_i$ sends the certificate to all before halting the protocol.

Deal:

1. On receiving $(\text{deal}, \text{sid}, \text{msg})$ from the environment such that sid is fresh, $\mathcal{P}_D$ does the following:
 - (a) Set $\text{dealmsg} = (\text{deal}, \text{sid}, \text{msg})$ and sign it:

$$\sigma^{\text{PKI-d}} \leftarrow \text{Sign}^{\text{PKI}}(\text{dealmsg})$$

- (b) Send $(\text{dealmsg}, \sigma^{\text{PKI-d}})$ to each $\{\mathcal{P}_i\}_{i \in [n]}$
This completes the first round.

2. Each party $\mathcal{P}_i$ does the following, ignoring any unsigned/malformed messages from the dealer:
 - (a) If some $(\text{dealmsg} = (\text{deal}, \text{sid}, \text{msg}), \sigma^{\text{PKI-d}})$ was received from the dealer, set $\text{echomsg}_i = (\text{echo}, \text{dealmsg}, \sigma^{\text{PKI-d}})$. Else, set $\text{echomsg}_i = (\text{echo}, \text{sid}, \text{nr})$.
 - (b) Set $\sigma_i^{\text{PKI}} \leftarrow \text{Sign}_{\text{sk}_i^{\text{PKI}}}^{\text{PKI}}(\text{echomsg}_i)$, and send $(\text{echomsg}_i, \sigma_i^{\text{PKI}})$ to all parties.

This completes the second round.

3. After collecting all messages $(\text{echomsg}_j, \sigma_j^{\text{PKI}})$ sent in the above round (ignoring malformed messages i.e. messages with non-verifying signatures), each $\mathcal{P}_i$ does the following:
 - If $\exists j, j' \in [n]$ such that $\text{echomsg}_j \neq \text{echomsg}_{j'}$ and both contain valid (but conflicting) dealmsg values, then set

$$\text{out}_i = \left(\text{sid}, \text{cheat}, \Omega_i = (\text{msg}_j, \text{msg}_{j'}, \sigma_j^{\text{PKI-d}}, \sigma_{j'}^{\text{PKI-d}}) \right)$$

where $(\text{msg}_j, \sigma_j^{\text{PKI-d}})$ and $(\text{msg}_{j'}, \sigma_{j'}^{\text{PKI-d}})$ are parsed from echomsg_j and $\text{echomsg}_{j'}$ respectively.

- Otherwise if $\exists \mathcal{J} \subseteq [n]$ such that $|\mathcal{J}| > t$ and $\text{echomsg}_j = (\text{echo}, \text{sid}, \text{nr})$ for each $j \in \mathcal{J}$, set

$$\text{out}_i = (\text{sid}, \text{nr}, \omega_i = \{\text{sid}, \text{echomsg}_j, \sigma_j^{\text{PKI}}\}_{j \in \mathcal{J}})$$

- Otherwise, set $\text{out}_i = (\text{sid}, \text{dealt}, \text{msg}_j, \sigma_j^{\text{PKI}})$ for any j where σ_j^{PKI} verifies.

Output out_i as computed above.

Transfer: On receiving $(\text{transfer}, \text{sid}, i, j)$ from the environment such that sid is fresh, $\mathcal{P}_i$ sends out_i to $\mathcal{P}_j$. $\mathcal{P}_j$ accepts out_i if any of the following hold:

- $\text{out}_i = (\text{sid}, \text{cheat}, \Omega_i)$ with Ω_i containing two different messages, both with valid signatures from $\mathcal{P}_D$.
- $\text{out}_i = (\text{nr}, \omega_i)$ with ω_i containing valid signatures by a set of at least $(t + 1)$ distinct parties on the message $(\text{echo}, \text{sid}, \text{nr})$.
- $\text{out}_i = (\text{sid}, \text{dealt}, \text{msg}, \sigma^{\text{PKI}})$, which contains a valid signed message of the dealer.

For convenience, we define an **audit** algorithm that takes as input a proof-of-cheating $\pi_\perp$ and outputs either $\{\text{accept}, \text{reject}\}$ indicating whether the proof was valid or not. We implicitly assume access to a PKI, which enables an external auditor running **audit** to verify authenticated messages and identify cheaters as well as that it is parametrized with the dealer's identity D . With respect to output of $\Pi_{\text{BC-PISA}}$, $\text{audit}_D(\pi_\perp)$ can be described as follows:

- Output **accept** if $\pi_\perp = (\text{sid}, \text{cheat}, \Omega_i)$ with Ω_i containing two different messages, both with valid signatures from $\mathcal{P}_D$ for the same sid .
- Output **accept** if $\pi_\perp = (\text{sid}, \text{nr}, \omega_i)$ with ω_i containing valid signatures by a set of at least $(t + 1)$ distinct parties on the message $(\text{echo}, \text{sid}, \text{nr})$.
- Else, output **reject**.

3 Preliminaries

3.1 Public Key Encryption

We take the definition of public key encryption directly from the work of Damgård *et al* [DRSY23b]. We additionally require our scheme to satisfy key verifiability, following the formalization in [KRSY26] (can be instantiated based on El Gamal encryption).

Definition 1 (Public-Key Encryption (PKE)). *A public-key encryption (PKE) scheme is a tuple of efficient algorithms $\text{PKE} = (\text{KeyGen}, \text{Enc}, \text{Dec})$ defined as follows.*

KeyGen $(1^\lambda) \rightarrow (\text{pk}, \text{sk})$: *The probabilistic algorithm KeyGen takes as input the security parameter $\lambda \in \mathbb{N}$, and outputs a public/secret key pair (pk, sk) .*

$\text{Enc}(\text{pk}, \text{msg}; \rho) \rightarrow \text{ct}$: The probabilistic algorithm Enc takes as input the public key pk , a message msg , and implicit randomness ρ , and outputs a ciphertext $\text{ct} = \text{Enc}(\text{pk}, \text{msg}; \rho)$.

$\text{Dec}(\text{sk}, \text{ct}) \rightarrow \text{msg}$: The deterministic algorithm Dec takes as input the secret key sk and a ciphertext ct and outputs $\text{msg} = \text{Dec}(\text{sk}, \text{ct})$ which is either equal to some message msg or to an error symbol $\perp$.

We require the following properties of a PKE scheme:

Correctness. We say that PKE satisfies correctness if for all $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda)$,

$$\Pr[\text{Dec}(\text{sk}, \text{Enc}(\text{pk}, \text{msg})) = \text{msg}] = 1$$

(where the randomness is taken over the internal coin tosses of algorithm Enc).

CPA Security. We say that PKE satisfies semantic security if for all PPT adversaries $\mathcal{A}$, for $(\text{msg}_0, \text{msg}_1) \leftarrow \mathcal{A}(1^\lambda)$, if $|\text{msg}_0| = |\text{msg}_1|$,

$$\Pr \left[\mathcal{A}(\text{pk}, \text{ct}) = b : \begin{array}{l} (\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda); b \leftarrow \{0, 1\}; \\ \text{ct} \leftarrow \text{Enc}(\text{pk}, \text{msg}_b) \end{array} \right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

(where the randomness is taken over the internal coin tosses of $\mathcal{A}$, KeyGen and Enc).

Key Verifiability. We say that PKE satisfies key verifiability if there exists a deterministic polynomial-time algorithm KeyVer such that:

- For all $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(1^\lambda)$,

$$\Pr[\text{KeyVer}(\text{pk}, \text{sk}) = 1] = 1.$$

- For all pk, sk such that (pk, sk) were not generated by $\text{KeyGen}(1^\lambda)$,

$$\Pr[\text{KeyVer}(\text{pk}, \text{sk}) = 1] \leq \text{negl}(\lambda).$$

3.2 Secret Sharing

Definition 2 (t -out-of- n secret sharing scheme). A t -out-of- n secret sharing scheme is a pair of algorithms $(\text{share}, \text{reconstruct}_t)$, such that:

- $(s_1, \dots, s_n) \leftarrow \text{share}(s, n, t; \rho)$ is a randomized algorithm that, on input a secret s , outputs a set of n shares $(s_1, \dots, s_n)$ using randomness ρ and fixes the threshold t .
- $s \leftarrow \text{reconstruct}_t(\{s_i\}_{i \in T})$ is a deterministic algorithm that, on input a set of shares $\{s_i\}_{i \in T}$, where $T \subseteq [n]$ and $|T| \geq t$, outputs the secret s or $\perp$ in case reconstruction fails.

A secret sharing scheme must satisfy the following properties:

Correctness: $\forall s, \forall T \subseteq [n]$ such that $|T| \geq t$, we have:

$$\Pr_{\rho} [\text{reconstruct}_t(\{s_i\}_{i \in T}) = s \mid (s_1, \dots, s_n) \leftarrow \text{share}(s, n, t; \rho)] = 1.$$

Perfect Security: $\forall s, s', \forall I \subseteq \{1, \dots, n\}$ where $|I| < t$, the following distributions are identical:

$$\begin{aligned} & \{(s_i : i \in I) \mid (s_1, \dots, s_n) \leftarrow \text{share}(s, n, t)\} \\ & \{(s'_i : i \in I) \mid (s'_1, \dots, s'_n) \leftarrow \text{share}(s', n, t)\}. \end{aligned}$$

We instantiate the secret sharing scheme using Shamir secret sharing.

Definition 3 (Shamir secret sharing scheme [Sha79]). Let $\mathbb{F}_q$ be a finite field with $q > n$.

- $\text{share}(s, n, t; \rho)$: Using randomness ρ , sample $a_1, \dots, a_{t-1} \xleftarrow{\rho} \mathbb{F}_q$, build the polynomial $p(x)$ as $p(x) = s + \sum_{i=1}^{t-1} a_i x^i$ and construct the shares $(s_1, \dots, s_n)$ as $s_i \leftarrow p(i)$.
- $\text{reconstruct}_t(\{s_i\}_{i \in T})$: On input a set of shares $\{s_i\}_{i \in T}$, where $T \subseteq [n]$ and $|T| \geq t$, check whether there exists a polynomial $p(x)$ of degree at most $t-1$ such that $p(i) = s_i$ for every $i \in T$. If such a polynomial exists, output $s = p(0)$; otherwise, output $\perp$.

3.3 Digital Signatures

Definition 4 (Digital signature scheme). A digital signature scheme is a tuple of efficient algorithms $(\text{KeyGen}, \text{Sign}, \text{Ver})$ defined as follows:

- $(\text{vk}, \text{sigk}) \leftarrow \text{KeyGen}(1^\lambda)$: The randomized key-generation algorithm KeyGen takes as input the security parameter and outputs a verification key vk and a signing key sigk .
- $\sigma \leftarrow \text{Sign}(\text{sigk}, \text{msg})$: The signing algorithm Sign takes as input a signing key sigk and a message msg and outputs a signature σ .
- $\{\text{accept}, \text{reject}\} \leftarrow \text{Ver}(\text{vk}, \text{msg}, \sigma)$: The verification algorithm Ver takes as input a verification key vk , a message msg , and a signature σ , and outputs accept if the signature is valid and reject otherwise.

We require the following properties of a digital signature scheme:

- **Correctness.** For every pair $(\text{vk}, \text{sigk}) \leftarrow \text{KeyGen}(1^\lambda)$ and every $\text{msg} \in \{0, 1\}^*$, we have that

$$\text{Ver}(\text{vk}, \text{msg}, \text{Sign}(\text{sigk}, \text{msg})) = \text{accept}.$$

- **Security.** For every PPT algorithm $\mathcal{A}$, we have that

$$\Pr \left[\mathcal{A}^{\text{Sign}(\text{sigk}, \cdot)} = (\text{msg}, \sigma) \text{ s.t. } \text{msg wasn't queried by } \mathcal{A} \text{ and } \text{Ver}(\text{vk}, \text{msg}, \sigma) = 1 \right] < \text{negl}(\lambda).$$

3.4 Garbling Scheme

Definition 5. [DRSY23b, KRSY26] A garbling scheme with independent key generation $\mathcal{G}$ is a pair of algorithms $(\text{KeyGen}, \text{Garble}, \text{Eval})$ defined as follows.

- $\mathbf{K} \leftarrow \text{KeyGen}(1^\lambda, \#\text{wires}, \rho^{\text{GC}})$: The algorithm KeyGen takes as input a unary representation of the security parameter λ and a number of input wires and outputs $\#\text{wires}$ pairs of garbled labels $\mathbf{K} = (K_0^1, K_1^1, \dots, K_0^{\#\text{wires}}, K_1^{\#\text{wires}})$. For simplicity, we assume that for every $i \in [\#\text{wires}]$ and $b \in \{0, 1\}$, it holds that $K_b^i \in \{0, 1\}^\lambda$.
- $\text{GC} \xleftarrow{\$} \text{Garble}(1^\lambda, C, \mathbf{K}, \rho^{\text{GC}})$: The algorithm Garble takes as input a unary representation of the security parameter λ , a Boolean circuit $C : \{0, 1\}^{\#\text{wires}} \rightarrow \{0, 1\}^m$ and keys $\mathbf{K}$, and outputs a garbled circuit GC .
- $y \leftarrow \text{Eval}(\text{GC}, K^1, \dots, K^{\#\text{wires}})$: The algorithm Eval takes as input the garbled circuit GC and $\#\text{wires}$ garbled labels $K^1, \dots, K^{\#\text{wires}}$, and outputs a value $y \in \{0, 1\}^m$.

We require the following properties of a garbling scheme:

- **Perfect Correctness.** We say $\mathcal{G}$ satisfies *perfect correctness* if for any Boolean circuit $C : \{0, 1\}^{\#\text{wires}} \rightarrow \{0, 1\}^m$ and $x = (x_1, \dots, x_{\#\text{wires}}) \in \{0, 1\}^{\#\text{wires}}$ it holds that

$$\Pr[\text{Eval}(\text{GC}, \mathbf{K}[x]) = C(x)] = 1,$$

where $\mathbf{K} \leftarrow \text{KeyGen}(1^\lambda, \#\text{wires}, \rho^{\text{KeyGen}})$ with $\mathbf{K} = (K^{0,1}, K^{1,1}, \dots, K^{0,\#\text{wires}}, K^{1,\#\text{wires}})$, $\text{GC} \leftarrow \text{Garble}(1^\lambda, C, \mathbf{K}, \rho^{\text{GC}})$, and $\mathbf{K}[x] := (K^{x_1,1}, \dots, K^{x_{\#\text{wires}},\#\text{wires}})$.

- **Privacy.** We say that $\mathcal{G}$ satisfies *privacy* if there exists a simulator simGC such that for every PPT adversary $\mathcal{A}$, it holds that

$$\Pr[\text{GC}^{\text{priv}} = 1] \leq \frac{1}{2} + \text{negl}(\lambda),$$

where GC^{priv} experiment is defined in the following figure:

The garbling scheme privacy experiment GC^{priv} .

1. $\mathcal{A}$ constructs a circuit $C : \{0, 1\}^{\#\text{wires}} \rightarrow \{0, 1\}^m$, outputs an input $x = (x_1, \dots, x_{\#\text{wires}}) \in \{0, 1\}^{\#\text{wires}}$ and sends them to the challenger.
2. The challenger chooses a random bit $b \in \{0, 1\}$ and does the following:
 - (a) If $b = 0$:
 - Sample labels $\mathbf{K} \leftarrow \text{KeyGen}(1^\lambda, \#\text{wires}, \rho^{\text{KeyGen}})$.
 - Sample garbled circuit $\text{GC} \leftarrow \text{Garble}(1^\lambda, C, \mathbf{K}, \rho^{\text{GC}})$.
 - Set $K^i := K^{x_i,i}$ for $i \in [\#\text{wires}]$.
 - (b) If $b = 1$:
 - Sample labels $\mathbf{K} \leftarrow \text{KeyGen}(1^\lambda, \#\text{wires}, \rho^{\text{KeyGen}})$, s.t. only one label is sampled for every wire.
 - Sample simulated garbling $\text{GC} \leftarrow \text{simGC}(C, C(x), \mathbf{K}, \rho^{\text{GC}})$.
 - Set $K^i := K^{x_i,i}$ for $i \in [\#\text{wires}]$.

3. The challenger sends $\mathbf{GC}, K^1, \dots, K^{\#\text{wires}}$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a bit b' .
5. $\mathcal{A}$ wins if $b = b'$, and we say that the experiment evaluates to 1 in this case.

3.5 Non-Interactive Zero-Knowledge Arguments of Knowledge

Definition 6 (Non-Interactive Zero-Knowledge Arguments of Knowledge). A non-interactive zero-knowledge argument of knowledge (NIZK) scheme is a tuple of efficient algorithms $\text{NIZK} = (\text{Gen}, \text{Prove}, \text{Verify}, \text{simP})$ defined as follows.

- $(\mathbf{crs}, \tau) \leftarrow \text{Gen}(1^\lambda, \mathcal{R})$: The algorithm Gen takes as input the security parameter $\lambda \in \mathbb{N}$ and a relation $\mathcal{R}$, and outputs the common reference string $\mathbf{crs}$ and the trapdoor τ for the NIZK system.
- $\pi \leftarrow \text{Prove}(\mathbf{crs}, \phi, w)$: The algorithm Prove takes as input the common reference string $\mathbf{crs}$ for a relation $\mathcal{R}$, a statement ϕ and a witness w , and outputs a proof π that $(\phi, w) \in \mathcal{R}$.
- $\{\text{accept}, \text{reject}\} \leftarrow \text{Verify}(\mathbf{crs}, \phi, \pi)$: The algorithm Verify takes as input the common reference string $\mathbf{crs}$ for a relation $\mathcal{R}$, a statement ϕ and a proof π , and verifies whether π proves the existence of a witness w such that $(\phi, w) \in \mathcal{R}$.
- $\pi \leftarrow \text{simP}(\mathbf{crs}, \tau, \phi)$: The algorithm simP takes as input the common reference string $\mathbf{crs}$ for a relation $\mathcal{R}$, the trapdoor τ and a statement ϕ , and outputs a simulated proof of the existence of a witness w such that $(\phi, w) \in \mathcal{R}$.

We require the following properties of a NIZK scheme:

Correctness. We say that NIZK satisfies correctness if for any $(\phi, w) \in \mathcal{R}$, we have that

$$\Pr[\text{Verify}(\mathbf{crs}, \phi, \pi) = \text{accept} \mid (\mathbf{crs}, \tau) \leftarrow \text{Gen}(1^\lambda, \mathcal{R}), \pi \leftarrow \text{Prove}(\mathbf{crs}, \phi, w)] \geq 1 - \text{negl}(\lambda).$$

Zero Knowledge. We say that NIZK satisfies zero knowledge if for all PPT adversaries $\mathcal{A}$,

$$\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + \text{negl}(\lambda)$$

in the following experiment:

The NIZK zero-knowledge experiment

1. The challenger computes $(\mathbf{crs}, \tau) \leftarrow \text{Gen}(1^\lambda, \mathcal{R})$, chooses $b \leftarrow \{0, 1\}$, and sends $\mathbf{crs}$ to $\mathcal{A}$.
2. Repeat $\text{poly}(\lambda)$ times:

- (a) $\mathcal{A}$ sends (ϕ, w) to the challenger.
- (b) If $b = 0$, compute $\pi \leftarrow \text{Prove}(\text{crs}, \phi, w)$.
- (c) If $b = 1$, compute $\pi \leftarrow \text{simP}(\text{crs}, \tau, \phi)$.
- (d) Send π to $\mathcal{A}$.
3. $\mathcal{A}$ outputs a bit b' .
4. $\mathcal{A}$ wins if $b = b'$.

Simulation Extractability. We say that NIZK satisfies simulation extractability if for all PPT adversaries $\mathcal{A}$ there exists an extraction algorithm $\text{simP.E}^{\mathcal{A}}$ such that

$$\Pr[\mathcal{A} \text{ wins}] \leq \text{negl}(\lambda)$$

in the following experiment:

The NIZK simulation-extractability experiment

1. The challenger computes $(\text{crs}, \tau) \leftarrow \text{Gen}(1^\lambda, \mathcal{R})$, sets $Q_{\text{sim}} = \emptyset$, and sends crs to $\mathcal{A}$.
2. Repeat $\text{poly}(\lambda)$ times:
 - (a) $\mathcal{A}$ sends a statement ϕ to the challenger.
 - (b) The challenger computes $\pi \leftarrow \text{simP}(\text{crs}, \tau, \phi)$, sends π to $\mathcal{A}$, and adds π to Q_{sim} .
3. $\mathcal{A}$ outputs (ϕ^*, π^*) .
4. Compute $w^* \leftarrow \text{E}(\tau, \phi^*, \pi^*)$.
5. $\mathcal{A}$ wins if all of the following hold:
 - $(\phi^*, w^*) \notin \mathcal{R}$,
 - $\text{Verify}(\text{crs}, \phi^*, \pi^*) = \text{accept}$,
 - $\pi^* \notin Q_{\text{sim}}$.

4 Impossibility of Two Round PISA With $n \leq 3t$

In this section, we prove that PISA cannot be achieved in two rounds without a broadcast channel, even if the parties hold arbitrary correlated randomness. (Our bound holds independently of whether broadcast is available in the second round; it only shows that broadcast is necessary in the first round.)

For simplicity, consider $t = 1$, $n = 3$. Consider a three-party functionality f where $\mathcal{P}_1$ has input $b \in \{2, 3\}$, $\mathcal{P}_2, \mathcal{P}_3$ have x_2, x_3 respectively, and $\mathcal{P}_1$ gets output x_b :

$$f(b, x_2, x_3) = (x_b, \perp, \perp).$$

Theorem 1 (Two-Round PISA Impossible for $n = 3$, $t = 1$). *There does not exist a two-round three-party protocol for f that achieves PISA security for a single corruption, in the absence of broadcast in the first round.*

In our proof, we make informal use of several properties of PISA protocols, which are implied by the ideal functionality described in Section 2.1: non-frameability, correctness and privacy. In the context of our function, privacy implies that a corrupt $\mathcal{P}_1$ should be unable to learn both x_2 and x_3 . This is clearly true in the ideal world, where $\mathcal{P}_1$ is only given one x_b by the ideal functionality.

Proof. For the sake of contradiction, assume that such a protocol does exist. We describe it in terms of the following functions: **first-msg** $_{i \rightarrow j}$ and **snd-msg** $_{i \rightarrow j}$ for $i, j \in \{1, 2, 3\}$, and **out** $_i$ for $i \in \{1, 2, 3\}$.

The function **first-msg** $_{i \rightarrow j}$ (respectively, **snd-msg** $_{i \rightarrow j}$) computes the first-round (respectively, second-round) message from $\mathcal{P}_i$ to $\mathcal{P}_j$. **first-msg** $_{i \rightarrow j}$ takes in $\mathcal{P}_i$'s input x_i and randomness ρ_i ; **snd-msg** $_{i \rightarrow j}$ additionally takes in whatever messages $\mathcal{P}_i$ received in the first round.

The function **out** $_i$ takes in $\mathcal{P}_i$'s input x_i and randomness ρ_i , together with all of the messages $\mathcal{P}_i$ received in both rounds; the entire protocol view of $\mathcal{P}_i$ can be recovered from these things. It outputs either the computed function output, or an identity j together with a proof of $\mathcal{P}_j$'s cheating. (We omit from our description the audit function that can be used to verify such an accusation.) Consider the following sequence of scenarios, also illustrated in Figure 1.

Scenario 3C, $\mathcal{P}_3$ corrupt: $\mathcal{P}_1$ uses input $b = 2$. Corrupt $\mathcal{P}_3$ does not send a second-round message to $\mathcal{P}_1$, and otherwise behaves honestly.

Scenario 1Cinp2, $\mathcal{P}_1$ corrupt: Corrupt $\mathcal{P}_1$ uses input $b = 2$, and pretends not to hear from $\mathcal{P}_3$ in the second round.

Scenario 2C, $\mathcal{P}_2$ corrupt: $\mathcal{P}_1$ uses input $b = 3$. Corrupt $\mathcal{P}_2$ does not send a second-round message to $\mathcal{P}_1$, and otherwise behaves honestly. Since $\mathcal{P}_1$ cannot produce a certificate of misbehavior, it must receive the correct output x_3 .

Scenario 1Cinp3, $\mathcal{P}_1$ corrupt: Corrupt $\mathcal{P}_1$ uses input $b = 3$, and pretends not to hear from $\mathcal{P}_2$ in the second round.

Scenario 1C, $\mathcal{P}_1$ corrupt: Corrupt $\mathcal{P}_1$ uses input $b = 2$ when talking to $\mathcal{P}_2$, and uses input $b = 3$ when talking to $\mathcal{P}_3$. $\mathcal{P}_1$ does not send a second-round message.

We can now make the following observations:

1. In Scenario 1Cinp2, by non-frameability, $\mathcal{P}_1$ cannot produce a certificate of misbehavior against the honest $\mathcal{P}_3$.
2. In Scenario 3C, the view of honest $\mathcal{P}_1$ is a subset of what corrupt $\mathcal{P}_1$ sees in Scenario 1Cinp2. So, honest $\mathcal{P}_1$ cannot produce a certificate of misbehavior against $\mathcal{P}_3$, and, by correctness, must instead receive the correct output x_2 .
3. In Scenario 1Cinp2, the view of corrupt $\mathcal{P}_1$ subsumes what honest $\mathcal{P}_1$ sees in Scenario 3C, so corrupt $\mathcal{P}_1$ learns x_2 .

In particular, $\mathcal{P}_1$ uses the following to learn x_2 :

- $\mathcal{P}_1$'s own input $b = 2$ and randomness ρ_1 ;
- First-round messages $\mathcal{P}_1$ receives, computed by the honest parties independently of b :

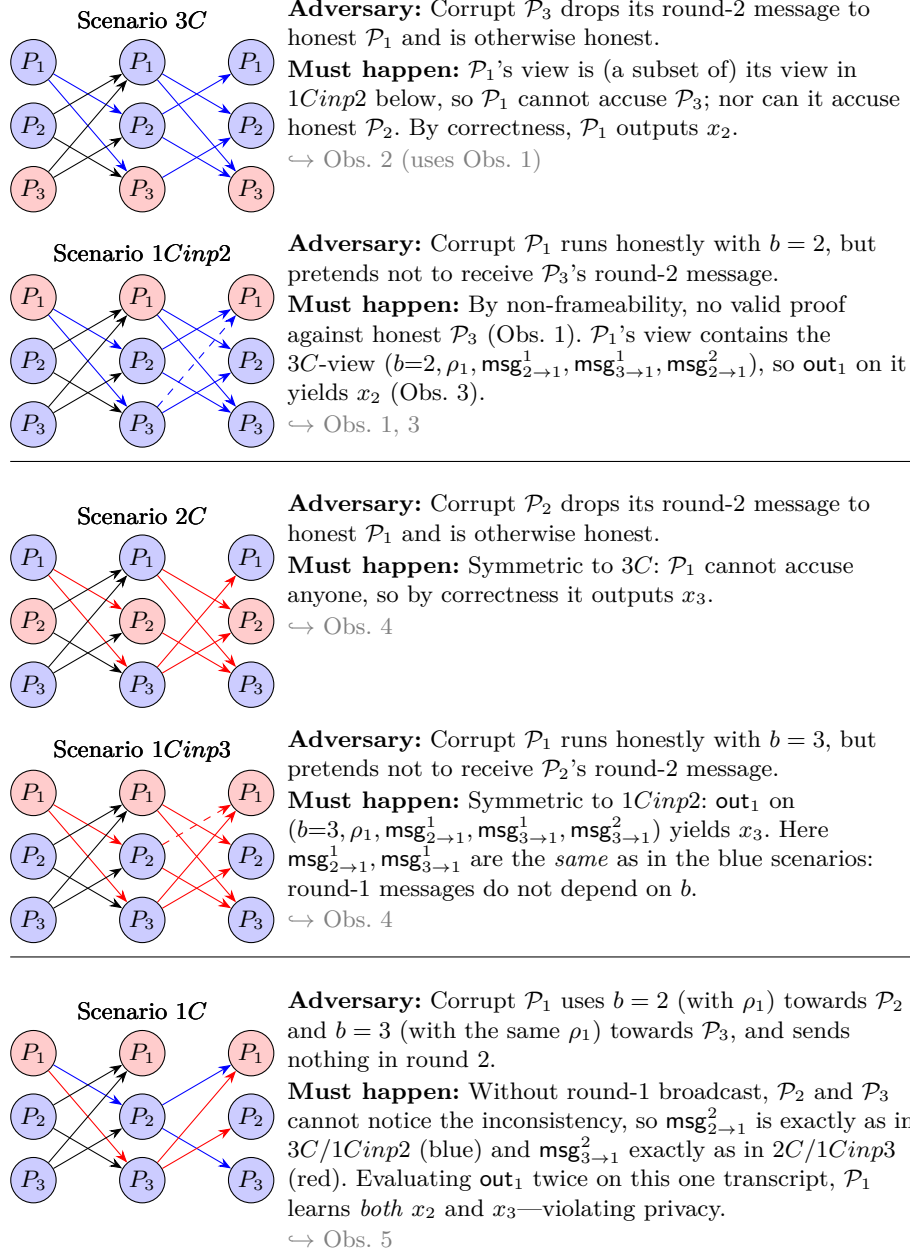


Fig. 1: The scenarios of the proof of Theorem 1, annotated with what security forces to happen in each, and the corresponding observation(s). Red node: corrupt party. Blue/red arrows depend on $\mathcal{P}_1$ using $b = 2/b = 3$, black arrows on neither. Dashed: receiver pretends not to have received it; missing: not sent.

- $\text{msg}_{2 \rightarrow 1}^1 = \text{frst-msg}_{2 \rightarrow 1}(x_2; \rho_2)$
- $\text{msg}_{3 \rightarrow 1}^1 = \text{frst-msg}_{3 \rightarrow 1}(x_3; \rho_3)$
- $\mathcal{P}_2$'s second-round message

$$\text{msg}_{2 \rightarrow 1}^2 = \text{snd-msg}_{2 \rightarrow 1}(x_2, \text{msg}_{1 \rightarrow 2}^1, \text{msg}_{3 \rightarrow 2}^1; \rho_2),$$

where $\text{msg}_{1 \rightarrow 2}^1 = \text{frst-msg}_{1 \rightarrow 2}(b = 2; \rho_1)$ and $\text{msg}_{3 \rightarrow 2}^1 = \text{frst-msg}_{3 \rightarrow 2}(x_3; \rho_3)$.

4. Applying a symmetric sequence of observations (on Scenarios 1Cinp3 and 2C), we can conclude that $\mathcal{P}_1$ can learn x_3 from *the same* first-round messages from $\mathcal{P}_2$ and $\mathcal{P}_3$, as well as a second-round message from $\mathcal{P}_3$:

$$\text{msg}_{3 \rightarrow 1}^2 = \text{snd-msg}_{3 \rightarrow 1}(x_3, \text{msg}_{1 \rightarrow 3}^1, \text{msg}_{2 \rightarrow 3}^1; \rho_3),$$

where $\text{msg}_{1 \rightarrow 3}^1 = \text{frst-msg}_{1 \rightarrow 3}(b = 3; \rho_1)$ and $\text{msg}_{2 \rightarrow 3}^1 = \text{frst-msg}_{2 \rightarrow 3}(x_2; \rho_2)$.

5. In Scenario 1C, $\mathcal{P}_1$ collects all of this information, and thus learns both x_2 and x_3 . This is a clear violation of privacy, thus proving our theorem.

Corollary 1 (Two-Round PISA Impossible for $n \leq 3t$). *Theorem 1 implies that there exist n -party functionalities f such that there does not exist a two-round n -party protocol for f that achieves PISA security for $t \geq \frac{n}{3}$, in the absence of broadcast in the first round.*

Corollary 1 follows by the standard logic: otherwise, a three-party protocol could be built by having each of the three parties execute the code of a third of the parties in the n -party protocol.

5 Three Round PISA With Honest Majority

In this section, we present a three-round PISA MPC protocol in the PKI model that is secure against $t < n/2$ corruptions. Our starting point is the compiler of [KR25], which transforms any r -round IA protocol using broadcast channels into a $2r + 2$ -round PISA protocol that relies only on point-to-point channels. Unfortunately, their compiler does not directly achieve our target round complexity of 3, since instantiating it with a round-optimal 2-round IA protocol [CGZ20] would still result in a 6-round protocol. We therefore investigate whether some of the rounds introduced by their compiler can be eliminated. At a high level, the compiler begins with a $2r$ -round phase in which each broadcast invocation in the original protocol is emulated using a two-round BC-PISA protocol from [KR25]. Informally, a BC-PISA protocol guarantees that each honest party either obtains the output of the broadcast or a proof-of-cheating certificate against the sender. We observe that if the underlying protocol invokes broadcast in fewer rounds, the overall round complexity of the compiled protocol is immediately reduced. Next, we observe that the additional rounds are required to generate proofs-of-cheating certificates for misbehaviors detected in the final (i.e., r -th) round of the underlying protocol. Thus, if the underlying protocol is such that aborts cannot be triggered by actions in its last round, this additional overhead can also be avoided.

Based on the above observations, we take as the underlying protocol Π the two-round protocol of [GLS15] in the PKI model, which achieves guaranteed output delivery, and exploit two key features of its design. First, the protocol can be modified to use broadcast only in the first round, as shown in [DMR⁺21]. Second, since the protocol already achieves guaranteed output delivery, aborts cannot occur due to misbehavior in the second round over point-to-point channels.

This leads to our three-round PISA construction Π_{PISA} : Every broadcast invocation in the first round of Π is replaced by an invocation of a BC-PISA protocol, which ensures that each honest party either receives the sender's message or obtains a proof-of-cheating certificate against the sender. There are two possibilities:

- (a) If all BC-PISA invocations succeed, the broadcast is perfectly emulated. The parties can thereby proceed to the second round of Π . By the guarantees of Π , all honest parties then obtain the output. The resulting protocol Π_{PISA} thus completes in three rounds: two rounds for the BC-PISA invocations, followed by a third round corresponding to the second round of Π .
- (b) Otherwise, if any honest party aborts during a BC-PISA invocation, it must have obtained a proof-of-cheating certificate. This can be forwarded to the remaining parties in the third round, preserving PISA guarantees.

We present the formal protocol Π_{PISA} in the $\mathcal{F}_{\text{BC-PISA}}$ -hybrid model, where $\mathcal{F}_{\text{BC-PISA}}$ (Section 2.2) is the ideal functionality for BC-PISA.

Protocol Π_{PISA} in the $\mathcal{F}_{\text{BC-PISA}}$ -hybrid model

Input: Each party $\mathcal{P}_i$ has input x_i

Building Blocks:

- Let $\mathcal{F}_{\text{BC}}$ denote the ideal functionality for broadcast. Π is a 2-round honest-majority protocol achieving guaranteed output delivery in the $\mathcal{F}_{\text{BC}}$ -hybrid model: The first round communication in Π involves invocations to $\mathcal{F}_{\text{BC}}$ (and does not have any point-to-point communication). The second round of Π involves only point-to-point communication (instantiated by [GLS15]).

The Protocol: Assuming the setup of Π , the protocol Π_{PISA} proceeds as follows:

- *Emulating Round 1 of Π :* Each party $\mathcal{P}_i$ does the following – Suppose Round 1 of Π requires $\mathcal{P}_i$ to broadcast a message msg (i.e. invoke $\mathcal{F}_{\text{BC}}$ as sender with message msg), then $\mathcal{P}_i$ invokes $\mathcal{F}_{\text{BC-PISA}}$ as sender with $(\text{deal}, \text{msg})$.
- *Emulating Round 2 of Π :* Each party $\mathcal{P}_i$ does the following:
 - If any of the above $\mathcal{F}_{\text{BC-PISA}}$ invocations terminated with a cheat ($\text{cert} = (\text{type}, j)$) against dealer $\mathcal{P}_j$, then $\mathcal{P}_i$ transfers this output

- to all other parties using $(\text{transfer}, i, k)$ for $k \in [n] \setminus i$ and terminates with output $\text{out}_i = (\text{type} \in \{\text{cheat}, \text{nr}\}, j)$ as appropriate.
 - Else, let b_j denote the output of the BC-PISA invocation with $\mathcal{P}_j$ as the sender. Let $(b_1, \dots, b_n)$ denote the first-round broadcast messages of Π . Compute and send Round 2 messages of Π over point-to-point channels.
- **Output:** Each P_i does the following – If any valid proof-of-cheating certificate ($\text{cert} = (\text{type}, j)$) was received as output of any $\mathcal{F}_{\text{BC-PISA}}$ invocations in the first step or via transfer in the second step, terminate with output $\text{out}_i = (\text{type} \in \{\text{cheat}, \text{nr}\}, j)$ identifying $\mathcal{P}_j$ as the cheater. Else, output the output of Π .

Round Complexity of Π_{PISA} . We note that if the invocations of $\mathcal{F}_{\text{BC-PISA}}$ are realized using the 2-round BC-PISA protocol of [KR25], and the **transfer** is realized in 1 round (as in the protocol of [KR25] by simply sending the proof-of-cheating certificate over point-to-point channels); then the overall protocol comprises of 3 rounds.

We state the theorem below. We assume the same PKI setup as in [GLS15], where the public keys are structured to enable threshold fully homomorphic encryption.

Theorem 2. *Assume a setup with CRS and (special) PKI (as in [GLS15]). Consider a 2-round protocol Π that uses only broadcast channels in the first round, only point-to-point channels in the second round and achieves guaranteed output delivery against a malicious adversary corrupting $t < n/2$ parties. Then, the Π_{PISA} realizes $\mathcal{F}_{\text{PISA}}$ in the $\mathcal{F}_{\text{BC-PISA}}$ -hybrid model, against a malicious adversary corrupting $t < n/2$ parties.*

Proof. Since there must exist a simulator $\mathcal{S}_{\Pi}$ (by virtue of security of Π), we can use $\mathcal{S}_{\Pi}$ to construct a simulator $\mathcal{S}$ for Π_{PISA} . To argue security, we show that any adversary $\mathcal{A}_{\text{PISA}}$ for Π_{PISA} can be transformed to an adversary $\mathcal{A}_{\Pi}$ for Π .

At a high level, the only difference between Π and Π_{PISA} is with respect to the first-round broadcast. In Π , the use of standard broadcast guarantees that all honest parties have a consistent view of every broadcast message. In contrast, in Π_{PISA} , an adversary $\mathcal{A}_{\text{PISA}}$ may abort a $\mathcal{F}_{\text{BC-PISA}}$ invocation in a way that causes some honest parties to receive the first-round message while others detect a cheat. Those parties that detect a cheat do not proceed to round 2 of Π . In this sense, Π_{PISA} is a partial emulation of Π : it faithfully runs Π for those honest parties that do not detect any misbehavior, while allowing others to terminate early with a proof-of-cheating. The reduction is therefore a matter of correctly filtering messages between the two executions of Π and Π_{PISA} . Below, we formalize this reduction where given a challenge execution of Π in the $\mathcal{F}_{\text{BC}}$

-hybrid model (either real or simulated by $\mathcal{S}_\Pi$), we construct an execution of Π_{PISA} where $\mathcal{A}_{\text{PISA}}$ attempts to decide if real or simulated. $\mathcal{A}_\Pi$ internally runs $\mathcal{A}_{\text{PISA}}$ and translates between the two executions as follows:

Case 1. $\mathcal{A}_{\text{PISA}}$ does not abort any $\mathcal{F}_{\text{BC-PISA}}$ invocation: $\mathcal{A}_\Pi$ behaves identical to $\mathcal{A}_{\text{PISA}}$: Each (deal, m) message sent to $\mathcal{F}_{\text{BC-PISA}}$ by a corrupt dealer $\mathcal{P}_j$ is forwarded as the corresponding $\mathcal{F}_{\text{BC}}$ (standard broadcast) invocation by $\mathcal{P}_j$ in Π . The second-round point-to-point messages are forwarded directly. In this case, the honest parties view in Π_{PISA} is identical to their view in Π : By the consistency property of $\mathcal{F}_{\text{BC-PISA}}$, they receive the same first-round broadcast message, proceed to compute their second-round messages and output exactly as in Π . This case is thus simulated perfectly by $\mathcal{S}_\Pi$.

Case 2. $\mathcal{A}_{\text{PISA}}$ aborts at least one $\mathcal{F}_{\text{BC-PISA}}$ invocation corresponding to a corrupt dealer, say, $\mathcal{P}_j$: In this case, if no honest party receives the intended message, $\mathcal{A}_\Pi$ broadcasts $\perp$ on behalf of corrupt $\mathcal{P}_j$ in the corresponding $\mathcal{F}_{\text{BC}}$ invocation in Π . Otherwise, it must be the case that at least one honest party receives a message, say m as output of $\mathcal{F}_{\text{BC-PISA}}$. Then, $\mathcal{A}_\Pi$ broadcasts m on behalf of corrupt $\mathcal{P}_j$ in the corresponding $\mathcal{F}_{\text{BC}}$ invocation in Π .

Next, $\mathcal{A}_\Pi$ filters messages sent by honest parties in round 2 of Π while forwarding them to $\mathcal{A}_{\text{PISA}}$ in Π_{PISA} : Only the messages of honest parties who did not discover a cheat in any of the first-round $\mathcal{F}_{\text{BC-PISA}}$ invocations are forwarded. On behalf of an honest party $\mathcal{P}_i$ who detected a cheat ($\text{cert} = (\text{type}, j)$) against dealer $\mathcal{P}_j$, $\mathcal{A}_\Pi$ forwards ($\text{cert} = (\text{type}, j)$) to $\mathcal{A}_{\text{PISA}}$, when $\mathcal{A}_{\text{PISA}}$ invokes $\mathcal{F}_{\text{BC-PISA}}$ with $(\text{transfer}, i, k)$ on behalf of a corrupt $\mathcal{P}_k$.

Lastly, $\mathcal{A}_\Pi$ outputs whatever $\mathcal{A}_{\text{PISA}}$ does.

This completes the description of $\mathcal{A}_\Pi$. We observe that any distinguishing advantage of $\mathcal{A}_{\text{PISA}}$ against Π_{PISA} translates directly to a distinguishing advantage of $\mathcal{A}_\Pi$ against Π . Since Π is secure, this advantage has to be negligible. For the honest parties' outputs, $\mathcal{S}$ instructs $\mathcal{F}_{\text{PISA}}$ to deliver to each honest party the certificate (type, j) it obtains from the emulated $\mathcal{F}_{\text{BC-PISA}}$, which always names a corrupt dealer, and y otherwise; the latter matches the real execution by the guaranteed output delivery of Π . We can thus conclude that Π_{PISA} securely realizes $\mathcal{F}_{\text{PISA}}$ in the $\mathcal{F}_{\text{BC-PISA}}$ -hybrid model.

6 Two Round PISA With Honest Super-majority

6.1 One-or-Nothing Secret Sharing with Omissions (1or0wo)

Looking ahead to our 2-round MPC protocol, we build on the technique of [DMR⁺21, DRSY23b]: Each party sends a garbled circuit that computes its second-round message, where the garbled circuit takes as input the set of first-round messages. The labels of these garbled circuits are distributed among the parties using a special type of secret sharing (referred to as *one-or-nothing secret sharing*). Since parties may receive inconsistent first-round messages (due to the

absence of broadcast), they vote on the version they received. There is also a provision to vote for $\perp$ if they do not receive a valid first-round message. The following guarantees must hold:

- At most one secret may be revealed by the sharing and the available ballots. Since the secret here refers to a label of a garbled circuit, revealing both labels would break the security of the garbling scheme.
- Whenever reconstruction fails, the MPC protocol must be able to provably identify a cheater.

Our setting introduces additional challenges compared to [DMR⁺21]. First, the absence of broadcast in both rounds, whereas in [DMR⁺21], broadcast is absent only in the first round. Second, we require cheater identification in the MPC protocol to be accompanied by a proof-of-cheating certificate. In [DMR⁺21], cheater identification in the second round is straightforward: since the second round uses broadcast, all parties can observe who failed to send a message or does not vote. In our setting, however, since second-round messages are sent point-to-point, a corrupt party can drop them (in particular, withhold its ballot), and this is a type of misbehavior that cannot by itself be proven to an external party.

We introduce a new variant of the one-or-nothing secret sharing scheme, which we call One-or-Nothing Secret Sharing with Omissions (**1or0wo**). As in [DMR⁺21], we define **1or0wo** as a semi-honest primitive and handle malicious behavior at the MPC protocol level. Looking ahead, the MPC protocol enforces correct sharings using zero-knowledge proofs and discards malformed ballots using an external validity check (in our construction, a ballot is a secret key that can be verified using **KeyVer**). Correctness of this primitive is therefore needed only for well-formed inputs. A check cannot, however, enforce that a party sends its ballot over point-to-point channels at all; we therefore allow for missing ballots in reconstruction.

In particular, an honest party can only count on receiving second-round messages from $n - t$ parties, as the t corrupt parties can drop their second-round messages without being held accountable. Among these received $n - t$ messages, it could be the case that only $n - 2t$ vote on the same value (representing the first-round message), while the remaining t vote on $\perp$ (claiming they did not receive the first-round message). This situation still requires robust reconstruction of the secret. This is because t parties voting on $\perp$ cannot be considered a proof against the sender: otherwise, t corrupt parties could produce such a certificate against an honest sender as well.

Accordingly, the reconstruction condition (which differs crucially from the one in [DMR⁺21, DRSY23b]) is stated as follows: if at least $n - 2t$ ballots vote for the same value v , all remaining available ballots vote for $\perp$, and there are at least $n - t$ ballots available in total, then the secret corresponding to v must be reconstructed.

Next, we analyze the desired privacy guarantee. A first requirement is immediate: *if fewer than $n - 2t$ parties vote for v , the secret for v must remain hidden*. However, this alone does not suffice. An adversarial sender could split

the honest parties into two sets of size $n - 3t$ receiving v and v' , respectively; together with the t corrupt parties, each set reaches $n - 2t$ votes, so both secrets would be reconstructed. This is possible whenever $2(n - 3t) \leq n - t$, i.e., for all $n \leq 5t$. We therefore additionally require that *the secret for v remains hidden whenever fewer than $n - t$ parties vote for v or $\perp$* . Intuitively, this means that more than t parties voted for some other value, and since at most t of them are corrupt, at least one *honest* party voted for a different value v' . In the MPC protocol, this corresponds to the sender (who determines the votes) having sent conflicting signed messages, leading to the sender being provably identified at the MPC level.

However, additional care must be taken to ensure that an adversarial sender cannot exploit $\perp$ votes. In particular, it should not be possible for an adversarial sender to split the *honest* parties into three groups of sizes $|\mathcal{S}_1| = n - 3t$, $|\mathcal{S}_2| = n - 3t$, and $|\mathcal{S}_3| = t$, where parties in $\mathcal{S}_1$ receive value v , parties in $\mathcal{S}_2$ receive value v' , and parties in $\mathcal{S}_3$ receive $\perp$. Parties in $\mathcal{S}_1$, together with t corrupt parties voting for v and $\mathcal{S}_3$ voting for $\perp$, would trigger reconstruction of v as this enables the adversary to satisfy the requirements of $n - 2t$ votes for v and at least $n - t$ votes on v or $\perp$. Symmetrically, parties in $\mathcal{S}_2$, together with t corrupt parties voting for v' and $\mathcal{S}_3$ voting for $\perp$ would trigger reconstruction of v' . Thus, we require

$$(n - 3t) + (n - 3t) + t > n - t \iff n > 4t.$$

While these two voting scenarios capture the main intuition, an adversary could also try other combinations of v , v' , and $\perp$ votes; our formal privacy definition addresses all of them, and $n > 4t$ turns out to suffice throughout.

Finally, we deliberately leave the case when valid ballots vote for different non- $\perp$ values outside the primitive and allow reconstruction to fail in such cases. As discussed above, such conflicts are resolved at the MPC level where the sender determining the votes is provably identified.

We are now ready to present the formal definition of One-or-Nothing Secret Sharing with Omissions. While the syntax closely follows the existing variants of one-or-nothing secret sharing [DMR⁺21, DRSY23b], the privacy and correctness guarantees are adapted to tolerate ballot omissions as described above.

Definition 7 (One-or-Nothing Secret Sharing with Omissions–1or0wo).

A *One-or-Nothing Secret Sharing with Omissions 1or0wo* = (**setup**, **share**, **vote**, **reconstruct**) is a protocol among n parties, where every party $\mathcal{P}_i$, for $i \in [n]$, plays the role of a voter, and one party additionally plays the role of the dealer $\mathcal{D}$. The algorithms have the following syntax; all of them are implicitly parameterized by n, t and the dealer's index $\mathcal{D}$.

- **setup**(1^λ) $\rightarrow$ (pk_i, sk_i) generates a voting key pair for a party.
- **share**($(x^1, \dots, x^\ell), \{\text{pk}_i\}_{i \in [n]}$) $\rightarrow$ **sh** is run by the dealer on its vector of secrets and outputs a sharing **sh**.
- **vote**(v_i, sk_i) $\rightarrow$ sh_i is run by each party on its vote $v_i \in [\ell] \cup \{\perp\}$ and outputs a ballot sh_i .

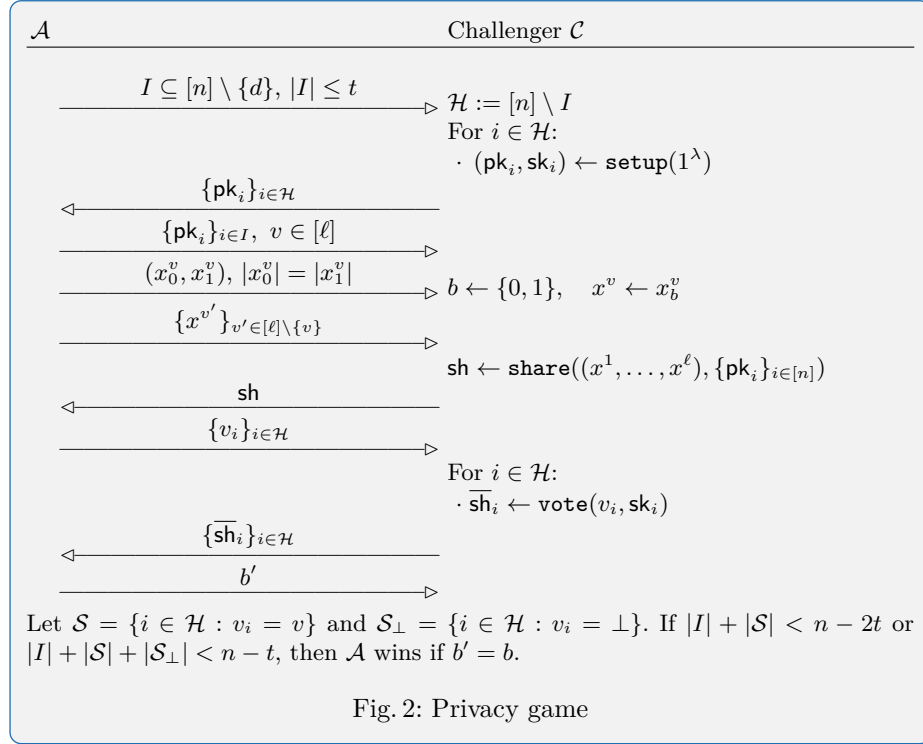
- **reconstruct**($\text{sh}, \{\overline{\text{sh}}_i\}_{i \in T}\}) \rightarrow \{x^v, \perp\}$ takes the sharing and the ballots of a subset $T \subseteq [n]$ of the parties and outputs either the secret x^v corresponding to a value v or $\perp$.⁴

We require that a **1or0wo** scheme satisfies privacy (definition 8) and correctness (definition 9),

Informally, privacy guarantees that if the dealer is honest, the adversary learns no information about x^v when either there are fewer than $n - 2t$ votes for v or fewer than $n - t$ votes for v and $\perp$, combined (counting corrupt parties' votes in both cases). Correctness requires that when at least $n - 2t$ parties produce their ballot using the same v , the other available ballots are produced with $\perp$, and at least $n - t$ ballots are available in total, then **reconstruct** returns x^v .

Definition 8 (One-or-Nothing Secret Sharing with Omissions: Privacy).

Informally, privacy guarantees, for an honest dealer, that the adversary learns no information about x^v when either there are fewer than $n - 2t$ votes for v or fewer than $n - t$ votes on v and $\perp$, combined (counting corrupt parties' votes in both cases). More formally, a One-or-Nothing Secret Sharing with Omissions scheme is private if, for every security parameter $\lambda \in \mathbb{N}$ and every PPT adversary $\mathcal{A}$, it holds that $\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + \text{negl}(\lambda)$ in the experiment in Figure 2.



⁴The cases in which $\perp$ is output are handled at the MPC level where a cheater is provably identified.

Definition 9 (One-or-Nothing Secret Sharing with Omissions: Correctness). *Informally, correctness requires that when at least $n - 2t$ parties produce their ballot using the same v , the other available ballots are produced with $\perp$, and at least $n - t$ ballots are available in total, then `reconstruct` returns x^v .*

More formally, a One-or-Nothing Secret Sharing with Omissions scheme is correct if for any security parameter $\lambda \in \mathbb{N}$, any vector of secrets $(x^1, \dots, x^\ell)$, any $v \in [\ell]$, and any disjoint $S, S_\perp \subseteq [n]$ with $|S| \geq n - 2t$ and $|S \cup S_\perp| \geq n - t$,

$$\Pr \left[\begin{array}{l} (\text{pk}_i, \text{sk}_i) \leftarrow \text{setup}(1^\lambda) \text{ for } i \in [n] \\ \text{sh} \leftarrow \text{share}((x^1, \dots, x^\ell), \{\text{pk}_i\}_{i \in [n]}) \\ \text{sh}_i \leftarrow \text{vote}(v, \text{sk}_i) \text{ for } i \in S \\ \text{sh}_i \leftarrow \text{vote}(\perp, \text{sk}_i) \text{ for } i \in S_\perp \\ x \leftarrow \text{reconstruct}(\text{sh}, \{\text{sh}_i\}_{i \in S \cup S_\perp}) \end{array} \right] \geq 1 - \text{negl}(\lambda),$$

where the probability is taken over the random coins of the algorithms. We say that the scheme is perfectly correct if this probability is 1, i.e., reconstruction succeeds for every choice of the random coins.

We note that privacy, as defined above, speaks about one position at a time: it hides x^v whenever the votes for v fall short of the reconstruction condition. A related question is how many positions can meet the condition simultaneously for a single set of ballots. The following lemma shows that, when $n > 4t$, at most one can, which is what makes the guarantee *one-or-nothing*.

Lemma 1 (One-or-nothing). *Let $n > 4t$. For every $I \subseteq [n]$ with $|I| \leq t$ and every choice of honest votes $\{v_i\}_{i \in \mathcal{H}}$, at most one position $v \in [\ell]$ satisfies both*

$$|I| + |\mathcal{S}_v| \geq n - 2t \quad \text{and} \quad |I| + |\mathcal{S}_v| + |\mathcal{S}_\perp| \geq n - t,$$

where $\mathcal{S}_v = \{i \in \mathcal{H} : v_i = v\}$ and $\mathcal{S}_\perp = \{i \in \mathcal{H} : v_i = \perp\}$. Consequently, in a private scheme all other positions remain hidden from the adversary.

Proof. Suppose that two distinct positions v_1 and v_2 both satisfy the two conditions; let $\mathcal{S}_1$ and $\mathcal{S}_2$ denote their sets of honest voters. Then

$$|I| + |\mathcal{S}_1| \geq n - 2t \quad \text{and} \quad |I| + |\mathcal{S}_2| + |\mathcal{S}_\perp| \geq n - t.$$

Since each honest party releases a key for at most one vote, these sets of honest voters are disjoint. Thus, $|\mathcal{S}_1| + |\mathcal{S}_2| + |\mathcal{S}_\perp| \leq n - |I|$, and adding the two inequalities gives

$$2n - 3t \leq 2|I| + |\mathcal{S}_1| + |\mathcal{S}_2| + |\mathcal{S}_\perp| \leq n + |I|.$$

It follows that $n \leq 3t + |I| \leq 4t$, contradicting $n > 4t$.

Construction. In this section, we describe our one-or-nothing secret sharing construction. Similar to the one-or-nothing secret sharing schemes of [DMR⁺21] [DRSY23b], our construction involves two layers of secret sharing. The dealer first secret-shares each secret x^v with threshold $n-t$, producing a first-level share for each party. Each first-level share is then re-shared with threshold $n-2t$, producing second-level shares.

During setup, each party generates an encryption key pair for each possible vote value, including $\perp$. The first-level share of a party is encrypted under that party's key for the corresponding position. The second-level shares are doubly encrypted: the share that $\mathcal{P}_j$ holds corresponding to $\mathcal{P}_i$'s first-level share is encrypted first under $\mathcal{P}_j$'s key for value v , and then under $\mathcal{P}_i$'s key for $\perp$. Opening this ciphertext requires both secret keys. The sharing is simply the set of these ciphertexts.

To vote, each party reveals the secret key corresponding to its vote. Reconstruction proceeds in two steps: First, the first-level shares of parties voting for some value v are recovered directly by decryption. For parties voting $\perp$, their first-level shares cannot be recovered directly; instead, the relevant second-level shares held by the v -voters are decrypted using both parties' revealed keys, and the first-level share is reconstructed from these. Finally, the secret is reconstructed using the available first-level shares.

For honestly generated sharings and ballots, reconstruction returns $\perp$ only if the available ballots contain conflicting non- $\perp$ votes, or if there are fewer than $n-2t$ available ballots for the same value or fewer than $n-t$ available ballots in total. These cases are handled at the MPC level.

One-or-Nothing Secret Sharing with Omissions

Building Blocks:

- Shamir secret sharing (`shamir.share`, `shamir.reconstruct`), as described in Section 3.2.
- A public-key encryption (`KeyGen`, `Enc`, `Dec`) with perfect correctness, as described in Section 3.1.

`setup`(1^λ) $\rightarrow$ (pk_i, sk_i)

1. Let $\text{pk}_i = (\text{pk}_i^1, \dots, \text{pk}_i^\ell, \text{pk}_i^\perp)$ and $\text{sk}_i = (\text{sk}_i^1, \dots, \text{sk}_i^\ell, \text{sk}_i^\perp)$, where $(\text{pk}_i^v, \text{sk}_i^v) \leftarrow \text{KeyGen}(1^\lambda)$ for $v \in [\ell] \cup \perp$.
2. Return $(\text{pk}_i, \text{sk}_i)$.

`share`($(x^1, \dots, x^\ell), \{\text{pk}_i\}_{i \in [n]}\rangle \rightarrow \text{sh}$:

1. For $v \in [\ell]$, compute $([x^v]_1, \dots, [x^v]_n) \leftarrow \text{shamir.share}(x^v, n, n-t; \rho^{x^v})$.
2. For $v \in [\ell]$ and $i \in [n]$, compute $([[x^v]_i]_1, \dots, [[x^v]_i]_n) \leftarrow \text{shamir.share}([x^v]_i, n, n-2t; \rho^{x^v})$.
3. For $v \in [\ell]$ and $i \in [n]$, compute $ct_i^v \leftarrow \text{Enc}(\text{pk}_i^v, [x^v]_i)$.
4. For $v \in [\ell]$ and $i, j \in [n]$, compute $ct_{i \rightarrow j}^v \leftarrow \text{Enc}(\text{pk}_i^\perp, \text{Enc}(\text{pk}_j^v, [[x^v]_i]_j))$.
5. Let $ct = (\{ct_i^v\}_{v \in [\ell], i \in [n]}, \{ct_{i \rightarrow j}^v\}_{v \in [\ell], i, j \in [n]})$.

```

6. Output  $\text{sh} = ct$ .

 $\text{vote}(v_i, \text{sk}_i) \rightarrow \overline{\text{sh}}_i$  : Output  $\overline{\text{sh}}_i = (v_i, \text{sk}_i^{v_i})$ .

 $\text{reconstruct}(\text{sh}, \{\overline{\text{sh}}_i\}_{i \in T}) \rightarrow \{x, \perp\}$  :

1. Parse  $\text{sh} = ct$ .
2. For each  $i \in T$ ,  $\overline{\text{sh}}_i = (v_i, \widehat{\text{sk}}_i)$ .
3. Let  $\mathcal{S} = \{i \in T : v_i \in [\ell]\}$  and  $\mathcal{S}_\perp = \{i \in T : v_i = \perp\}$ . If there exist  $i, j \in \mathcal{S}$  with  $v_i \neq v_j$ , output  $\perp$ . If  $|\mathcal{S}| < n - 2t$  or  $|\mathcal{S} \cup \mathcal{S}_\perp| < n - t$ , output  $\perp$ . Let  $v$  denote the common non- $\perp$  vote.
4. For  $i \in \mathcal{S}$ , compute  $x_i \leftarrow \text{Dec}(\widehat{\text{sk}}_i, ct_i^v)$  and output  $\perp$  if decryption fails.
5. For  $i \in \mathcal{S}_\perp$  and  $j \in \mathcal{S}$ , compute  $x_{i \rightarrow j} \leftarrow \text{Dec}(\widehat{\text{sk}}_j, \text{Dec}(\widehat{\text{sk}}_i, ct_{i \rightarrow j}^v))$  and output  $\perp$  if decryption fails.
6. For  $i \in \mathcal{S}_\perp$ , compute  $x_i \leftarrow \text{shamir.reconstruct}_{n-2t}(\{(j, x_{i \rightarrow j})\}_{j \in \mathcal{S}})$  and output  $\perp$  if reconstruction fails.
7. Compute  $x \leftarrow \text{shamir.reconstruct}_{n-t}(\{(i, x_i)\}_{i \in \mathcal{S} \cup \mathcal{S}_\perp})$  and output  $\perp$  if reconstruction fails. Otherwise output  $x$ .

```

Fig. 3: One-or-Nothing Secret Sharing with Omissions

Theorem 3. *The One-or-Nothing Secret Sharing with Omissions scheme construction in Figure 3 is private, as long as $(\text{KeyGen}, \text{Enc}, \text{Dec})$ is a secure public key encryption scheme (Section 3.1) for $n = O(\log(\lambda))$.*

Proof. Let $\mathcal{H}$ and $\mathcal{I}$ denote the sets of honest and corrupt parties, respectively. Let $\mathcal{S} = \{i \in \mathcal{H} : v_i = v\}$, $\overline{\mathcal{S}} = \mathcal{H} \setminus \mathcal{S}$, and $\mathcal{S}_\perp = \{i \in \mathcal{H} : v_i = \perp\}$. The winning condition requires

$$|\mathcal{I}| + |\mathcal{S}| < n - 2t \quad \text{or} \quad |\mathcal{I}| + |\mathcal{S}| + |\mathcal{S}_\perp| < n - t.$$

We consider both cases below, since otherwise the adversary does not win.

Recall that the dealer is honest in this case. Let us first unpack the information that the adversary learns:

1. From sh , it learns the ciphertexts $\{ct_i^{v'}\}_{v' \in [\ell], i \in [n]}$, which encrypt the first-level shares of each secret under the corresponding party's public key.
2. From sh , it also learns the ciphertexts $\{ct_{i \rightarrow j}^{v'}\}_{v' \in [\ell], i, j \in [n]}$, which encrypt the reshares under the original share holder's public key for $\perp$ and the reshare holder's public key for position v' .
3. From $\{\overline{\text{sh}}_i\}_{i \in \mathcal{S}}$, it learns the private keys $\{\text{sk}_i^v\}_{i \in \mathcal{S}}$.
4. From $\{\overline{\text{sh}}_i\}_{i \in \overline{\mathcal{S}}}$, it learns the private keys $\{\text{sk}_i^{v_i}\}_{i \in \overline{\mathcal{S}}}$.

We now argue that the above information does not give the adversary more than a negligible advantage in guessing which of the two challenge values was shared at position v while satisfying the winning condition.

Below, we give the argument for a fixed choice of $\mathcal{S}$ and $\mathcal{S}_\perp$ ⁵. Among the honest parties, the adversary learns a secret key corresponding to position v only from the parties in $\mathcal{S}$, and it knows all secret keys of corrupt parties. From the ciphertexts $\{ct_i^v\}_{i \in [n]}$, the adversary can therefore obtain at most $|\mathcal{I}| + |\mathcal{S}|$ first-level shares of x_b^v . The remaining ciphertexts encrypting the first-level shares are encrypted under honest parties' public keys whose corresponding secret keys are not revealed and are therefore protected by the CPA security of the encryption scheme.

Suppose $|\mathcal{I}| + |\mathcal{S}| < n - 2t$, i.e. $|\mathcal{I}| + |\mathcal{S}| \leq n - 2t - 1$. Consider the resharing of a first-level share $[x^v]_i$. Opening both layers of the ciphertext $ct_{i \rightarrow j}^v = \text{Enc}(\text{pk}_i^\perp, \text{Enc}(\text{pk}_j^v, [[x^v]_i]_j))$ requires both $\text{sk}_i^\perp$ and sk_j^v ; otherwise, at least one layer is protected by CPA security. In particular, decrypting the inner layer requires sk_j^v . As argued above, such a key is available for at most $|\mathcal{I}| + |\mathcal{S}| \leq n - 2t - 1$ parties, counting every corrupt party. Consequently, for any first-level share $[x^v]_i$, the adversary obtains at most $n - 2t - 1$ of its reshares. Since every first-level share is secret shared with reconstruction threshold $n - 2t$ using independent randomness, these reshares jointly reveal no information about the corresponding first-level share. It thus follows that the adversary does not learn any additional first-level share and has at most $|\mathcal{I}| + |\mathcal{S}| \leq n - 2t - 1$ first-level shares of x_b^v .

Now suppose that $|\mathcal{I}| + |\mathcal{S}| \geq n - 2t$. The winning condition then requires $|\mathcal{I}| + |\mathcal{S}| + |\mathcal{S}_\perp| < n - t$. In addition to the first-level shares indexed by $\mathcal{I} \cup \mathcal{S}$ that it obtains directly, the adversary may reconstruct the first-level shares indexed by $\mathcal{I} \cup \mathcal{S}_\perp$ using their reshares. For every other first-level share, the outer encryption of its reshares uses an honest party's key for $\perp$ that is not revealed, and is therefore protected by CPA security. Thus, the adversary learns at most

$$|\mathcal{I} \cup \mathcal{S} \cup \mathcal{S}_\perp| = |\mathcal{I}| + |\mathcal{S}| + |\mathcal{S}_\perp| < n - t$$

first-level shares. We note that the reshares of these first-level shares reveal no information about the remaining first-level shares, since the re-sharings use independent randomness.

Further, we note that the choices of the parties in $\bar{\mathcal{S}}$ cannot increase the number of secret keys corresponding to position v known to the adversary. If such a party votes for $\perp$, the adversary learns only its secret key for $\perp$, while if it votes for another position $v' \neq v$, the adversary learns only its secret key for v' . In neither case does it learn the party's secret key for position v .

In either case, the adversary learns fewer than $n - t$ first-level shares of x_b^v . Since x_b^v is secret shared with reconstruction threshold $n - t$, privacy of the

⁵Honest votes are chosen after sh , but a CPA reduction must place the challenge during sharing. The reduction therefore guesses each honest party's vote (v , $\perp$, or other), embeds the challenge in the ciphertexts whose keys stay hidden under that guess, and outputs a random bit if the guess turns out wrong. Since sh is independent of the votes, the guess is right with probability $3^{-|\mathcal{H}|}$, and conditioned on a correct guess the game is simulated perfectly; the reduction thus loses a factor 3^n , which is polynomial in λ for $n = O(\log \lambda)$.

underlying secret sharing scheme implies that these shares reveal no information about whether x_0^v or x_1^v was shared.

Together with CPA security, this bounds the probability that the adversary satisfies one of the two inequalities and guesses b correctly. Hence, $\Pr[\mathcal{A} \text{ wins}] \leq \frac{1}{2} + \text{negl}(\lambda)$; completing the proof.

Theorem 4. *The construction in Figure 3 is perfectly correct, as long as Shamir secret sharing (Section 3.2) and (KeyGen, Enc, Dec) (Section 3.1) satisfy perfect correctness.*

Proof. Let $\mathcal{S}$ and $\mathcal{S}_\perp$ be the sets of parties voting for v and for $\perp$, respectively, with $|\mathcal{S}| \geq n - 2t$ and $|\mathcal{S} \cup \mathcal{S}_\perp| \geq n - t$, and consider any execution in which the keys, the sharing and the ballots are generated honestly (with arbitrary random coins). We show that **reconstruct** outputs x^v .

Since all ballots in $\mathcal{S}$ vote for v and all remaining available ballots vote for $\perp$, there are no conflicting votes, and the two size conditions are satisfied by assumption. Hence, **reconstruct** does not output $\perp$ in step 3 and correctly identifies v as the common non- $\perp$ vote.

Next, we argue that every decryption succeeds. For $i \in \mathcal{S}$, the ballot $\overline{\text{sh}}_i$ reveals sk_i^v , which is the secret key under which $ct_i^v = \text{Enc}(\text{pk}_i^v, [x^v]_i)$ was computed; by correctness of the encryption scheme, decryption yields $[x^v]_i$. For $i \in \mathcal{S}_\perp$ and $j \in \mathcal{S}$, the ballots reveal $\text{sk}_i^\perp$ and sk_j^v , which are exactly the keys of the two encryption layers of $ct_{i \rightarrow j}^v$; by correctness of the encryption scheme, decrypting both layers yields $[x^v]_i$.

It remains to argue that both levels of Shamir reconstruction succeed. For every $i \in \mathcal{S}_\perp$, the decrypted values $\{[x^v]_i\}_{j \in \mathcal{S}}$ are $|\mathcal{S}| \geq n - 2t$ shares of $[x^v]_i$, which was shared with reconstruction threshold $n - 2t$. By correctness of Shamir secret sharing, they reconstruct to $[x^v]_i$. Together with the first-level shares of the parties in $\mathcal{S}$, **reconstruct** thus holds the first-level shares $\{[x^v]_i\}_{i \in \mathcal{S} \cup \mathcal{S}_\perp}$, that is, at least $n - t$ shares of x^v , which was shared with reconstruction threshold $n - t$. By correctness of Shamir secret sharing, these reconstruct to x^v , which **reconstruct** outputs.

Thus, perfect correctness of the construction in Figure 3 follows directly from perfect correctness of the encryption scheme and of Shamir secret sharing.

6.2 MPC Protocol

In this section we present our two-round PISA protocol $\Pi_{\text{PISA-2}}$ for $n > 4t$. The protocol is a compiler: it takes a two-round protocol Π_{GOD} with guaranteed output delivery in the broadcast model and turns it into a two-round protocol over point-to-point channels in which every honest party either obtains the output of Π_{GOD} or a publicly verifiable certificate against a corrupt party.

The underlying protocol Π_{GOD} . We write Π_{GOD} as a tuple of algorithms $\{\text{frst-msg}_i, \text{snd-msg}_i, \text{out}\}_{i \in [n]}$. $\mathcal{P}_i$ computes its first-round message as $\text{msg}_i^1 = \text{frst-msg}_i(x_i, \rho_i)$, its second-round message as $\text{msg}_i^2 = \text{snd-msg}_i(x_i, \rho_i, \{\text{msg}_j^1\}_{j \in [n]})$, and the

output as $y = \text{out}(\{\text{msg}_j^1\}_j, \{\text{msg}_j^2\}_j)$. We assume that Π_{GOD} has several properties.

1. *Broadcast-only.* For simplicity, we assume that every message is addressed to all parties, so that a party's round- r message can be represented as a single string. Private communication, where needed, can be realized by encrypting to the recipient's public encryption key (available as part of setup).
2. *Straight-line simulation.* The simulator of Π_{GOD} never needs to rewind the adversary. This lets us build a similarly straight-line simulator for $\Pi_{\text{PISA-2}}$.
3. *First-round input extraction.* The simulator of Π_{GOD} can extract the corrupt parties' inputs from their first-round messages alone. This allows our $\Pi_{\text{PISA-2}}$ simulator to use the Π_{GOD} simulator to extract the corrupt parties' inputs in time to obtain and program in the output, even if corrupt parties are rushing.
4. *Public output.* out is computed from the public transcript $(\{\text{msg}_j^1\}, \{\text{msg}_j^2\})$ only, without any party's private state.

The protocol of Gordon *et al.* [GLS15], as used in [DMR⁺21], has all the above properties: its first round consists of encryptions and its second round of threshold decryption shares, from which the output is publicly reconstructed. The setup of Π_{GOD} is included explicitly in our setup, alongside the keys of our own building blocks and a crs for the NIZK.

Overview of $\Pi_{\text{PISA-2}}$. In the first round, $\mathcal{P}_i$ sends msg_i^1 from the underlying protocol Π_{GOD} . In Π_{GOD} , msg_i^1 would be sent over broadcast; in the absence of broadcast, we instead send it using the two-round $\Pi_{\text{BC-PISA}}$ protocol. $\Pi_{\text{BC-PISA}}$ ensures that after two rounds, each honest party either holds the (signed) broadcast message or a certificate of cheating. However, the issue we face is that honest parties must send their second-round messages *before* $\Pi_{\text{BC-PISA}}$ has time to complete.

To solve this issue, we follow the garbled-circuit paradigm of [DMR⁺21]. Let C_i be the public circuit computing snd-msg_i ; its input wires carry (x_i, ρ_i) followed by the n first-round messages. In the first round, $\mathcal{P}_i$ sends a garbling of C_i and its input labels: the labels for the wires carrying $\mathcal{P}_i$'s own input and randomness are sent in the clear, since $\mathcal{P}_i$ knows these values, whereas for every wire carrying a bit of some msg_j^1 the pair of labels $(K^{\alpha,0}, K^{\alpha,1})$ is distributed using our one-or-nothing secret sharing scheme 1or0wo (Figure 3). To ensure that garbling and sharing is correct, $\mathcal{P}_i$ must include a zero knowledge proof of correct garbling and sharing. $\mathcal{P}_i$ cannot know at this point which version of msg_j^1 the other parties will receive; the garbled circuit lets each evaluator compute snd-msg_i on whatever version is eventually agreed upon, without any further participation of the garbler. In fact, $\mathcal{P}_i$ sends its garbling, the label sharings and the zero-knowledge proof through $\Pi_{\text{BC-PISA}}$ as well.

In the second round, every party votes. For each wire α carrying a bit of msg_j^1 , $\mathcal{P}_i$ releases the voting key for the bit it received, or for $\perp$ if it received no valid first-round message from $\mathcal{P}_j$. At the end of the second round, each party collects the ballots it has received, reconstructs one label per input wire

of every garbled circuit, evaluates all garbled circuits to obtain $\{\text{msg}_j^2\}_{j \in [n]}$, and outputs $\text{out}(\{\text{msg}_j^1\}, \{\text{msg}_j^2\})$. This might fail in several ways. If a party cannot recover the output, it instead outputs one of several kinds of certificate of cheating. Different honest parties may hold different views and may thus end with different verdicts; this is exactly the selective abort permitted by PISA.

Remark 1 (Delayed Receipt of Garbled Circuits). The garbled circuit is Shamir-shared rather than sent in the clear. Instead of including GC_i in its first-round message, $\mathcal{P}_i$ Shamir-shares it, and it is only reconstructed in the second round. The circuit is thus fixed in round one, but only becomes available in round two. Looking ahead, this is what allows the simulator to garble honest parties' circuits *after* the adversary's first-round messages are fixed and their inputs extracted, so that the simulated circuit can be programmed to output the simulated second-round message corresponding to the ideal output (Section 6.3).

Kinds of Failure. There are several kinds of failure — associated certificates of cheating — that $\mathcal{P}_i$ might end up with.

1. $\Pi_{\text{BC-PISA}}$ might have failed; $\mathcal{P}_i$ then obtains a certificate of cheating from $\Pi_{\text{BC-PISA}}$.
2. The zero knowledge proof of correct garbling and sharing might have failed; the signed incorrect proof is then a certificate of cheating.
3. Reconstruction of a garbled circuit might have failed; if so, the signed shares implicate the garbler.

Notation. Let $C_i(x_i, \rho_i, \{\text{msg}_j^1\}_{j \in [n]})$ denote the public Boolean circuit computing snd-msg_i , as described above. For simplicity, we assume that (x_i, ρ_i) consists of $|\text{inp}|$ bits, and each first-round message is $|\text{msg}|$ bits long, so each circuit has $\#\text{wires} = |\text{inp}| + n \cdot |\text{msg}|$ input bits. We assume that each C_i is of the same size and is public.

Let C_i have the following order:

1. Input wires on positions $[|\text{inp}|]$ correspond to $\mathcal{P}_i$'s input x_i and randomness ρ_i .
2. For $j \in [n]$, positions $\{|\text{inp}| + (j-1)|\text{msg}| + \beta \mid \beta \in [|\text{msg}|]\}$ correspond to msg_j^1 .

We use binary `1or0wo`, with positions 0 and 1. The keys $\text{pk}_j^{i,\alpha}$ and $\text{sk}_j^{i,\alpha}$ denote the corresponding key tuples for 0, 1, $\perp$. Party $\mathcal{P}_i$ generates its keys and registers the public keys before round 1. Voting keys are used only for their one label-sharing instance.

Relation for NIZK We use a NIZK for the following relation. The auxiliary coins ρ_i^{aux} contain the randomness for garbling, sharing, encryption, and signatures. This randomness is implicit in invocations of garbling, sharing, encryption and signing in Figure 4 (we suppress it from the notation).

$$\mathcal{R} = \left\{ \begin{array}{l} \phi = \left(i, \text{msg}_i^1, \{ct_j^{\text{GC}_i}\}_{j \in [n]}, \{K_i^\alpha\}_{\alpha \in [|\text{inp}|]}, \{\text{sh}_i^\alpha\}_{\alpha = |\text{inp}|+1}^{\#\text{wires}} \right) \\ w = ((x_i, \rho_i), \text{sigk}_i, \rho_i^{\text{aux}}) \end{array} \right\} \left\{ \begin{array}{l} \text{msg}_i^1 = \text{fst-msg}_i(x_i, \rho_i), \\ \mathbf{K}_i = \text{GS.KeyGen}(1^\lambda, \#\text{wires}), \\ \text{GC}_i = \text{Garble}(1^\lambda, C_i, \mathbf{K}_i). \\ \{[\text{GC}_i]_j\}_{j \in [n]} = \text{shamir.share}(\text{GC}_i, n, 2t+1), \\ \forall j \in [n]: \\ \quad \sigma_j^{\text{GC}_i} = \text{DS.Sign}(\text{sigk}_i, (j, [\text{GC}_i]_j)), \\ \quad \text{DS.Ver}(\text{vk}_i, (j, [\text{GC}_i]_j), \sigma_j^{\text{GC}_i}) = \text{accept}, \\ \quad ct_j^{\text{GC}_i} = \text{Enc}(\text{pk}_j^{\text{GC}}, ([\text{GC}_i]_j, \sigma_j^{\text{GC}_i})), \\ \forall \alpha \in [|\text{inp}|]: \quad K_i^\alpha = K_i^{\alpha, (x_i \parallel \rho_i)[\alpha]}, \\ \forall \alpha \in \{|\text{inp}|+1, \dots, \#\text{wires}\}: \\ \quad \text{sh}_i^\alpha = \text{1or0wo.share}((K_i^{\alpha,0}, K_i^{\alpha,1}), \\ \quad \quad \{\text{pk}_j^{i,\alpha}\}_{j \in [n]}) \end{array} \right\}$$

PISA Protocol $\Pi_{\text{PISA}-2}$

We assume $n > 4t$, with $n = O(\lambda)$.

Input: Each party $\mathcal{P}_i$ has input x_i , randomness ρ_i for Π_{GOD} , and independent auxiliary randomness ρ_i^{aux} .

Building Blocks:

- The One-or-Nothing Secret Sharing with Omissions scheme **1or0wo** = (**setup**, **share**, **vote**, **reconstruct**) (Section 6).
- A two-round BC-PISA protocol $\Pi_{\text{BC-PISA}}$ (Section 2.2).
- A two-round GOD protocol Π_{GOD} for f , represented by $\{\text{fst-msg}_i, \text{snd-msg}_i, \text{out}\}_{i \in [n]}$, with first-round input extraction.
- A NIZK scheme (**NIZK.Gen**, **NIZK.Prove**, **NIZK.Verify**) (Section 3.5).
- A secure garbling scheme (**GS.KeyGen**, **Garble**, **Eval**) with independent label generation (Section 3.4).
- A Shamir secret sharing scheme (**shamir.share**, **shamir.reconstruct**) (Section 3.2).
- A secure key-verifiable PKE scheme (**PKE.KeyGen**, **Enc**, **Dec**, **KeyVer**) (Section 3.1).
- A secure digital signature scheme (**DS.KeyGen**, **DS.Sign**, **DS.Ver**) (Section 3.3).

Setup: We assume the parties have access to the following setup:

- The prescribed public and private setup of Π_{GOD} .
- Independent voting keys $(\text{pk}_i^{j,\alpha,b}, \text{sk}_i^{j,\alpha,b}) \leftarrow \text{PKE.KeyGen}(1^\lambda)$ for $i, j \in [n]$, $\alpha \in \{|\text{inp}|+1, \dots, \#\text{wires}\}$, and $b \in \{0, 1, \perp\}$, where $\mathcal{P}_i$ holds the secret key. We let $\text{pk}_i^{j,\alpha} = (\text{pk}_i^{\alpha,0}, \text{pk}_i^{\alpha,1}, \text{pk}_i^{\alpha,\perp})$ and $\text{sk}_i^{j,\alpha} = (\text{sk}_i^{\alpha,0}, \text{sk}_i^{\alpha,1}, \text{sk}_i^{\alpha,\perp})$.
- Independent $(\text{pk}_i^{\text{GC}}, \text{sk}_i^{\text{GC}}) \leftarrow \text{PKE.KeyGen}(1^\lambda)$ for $i \in [n]$.
- $(\text{vk}_i, \text{sigk}_i) \leftarrow \text{DS.KeyGen}(1^\lambda)$ for $i \in [n]$.
- A trusted **crs** for $\mathcal{R}$.

Round 1: Each party $\mathcal{P}_i$ does the following:

- Compute $\text{msg}_i^1 \leftarrow \text{fst-msg}_i(x_i, \rho_i)$.

- Compute $\mathbf{K}_i \leftarrow \text{GS.KeyGen}(1^\lambda, \#\text{wires})$ and $\text{GC}_i \xleftarrow{\$} \text{Garble}(C_i, \mathbf{K}_i)$.
- Compute $\{[\text{GC}_i]_j\}_{j \in [n]} \leftarrow \text{shamir.share}(\text{GC}_i, n, 2t + 1)$.
- For $j \in [n]$, compute $\sigma_j^{\text{GC}_i} \leftarrow \text{DS.Sign}(\text{sigk}_i, (j, [\text{GC}_i]_j))$ and $ct_j^{\text{GC}_i} \leftarrow \text{Enc}(\text{pk}_j^{\text{GC}}, ([\text{GC}_i]_j, \sigma_j^{\text{GC}_i}))$.
- Set $K_i^\alpha = K_i^{\alpha, (x_i \parallel \rho_i)[\alpha]}$ for $\alpha \in [|\text{inp}|]$.
- For $j \in [n]$ and $\beta \in [|\text{msg}|]$, let $\alpha = |\text{inp}| + (j - 1)|\text{msg}| + \beta$, and compute $\text{sh}_i^\alpha \leftarrow \text{1or0wo.share}((K_i^{\alpha, 0}, K_i^{\alpha, 1}), \{\text{pk}_j^{i, \alpha}\}_{j \in [n]})$.
- Set $\phi_i = (i, \text{msg}_i^1, \{ct_j^{\text{GC}_i}\}_{j \in [n]}, \{K_i^\alpha\}_{\alpha \in [|\text{inp}|]}, \{\text{sh}_i^\alpha\}_{\alpha = |\text{inp}| + 1}^{\#\text{wires}})$ and $w_i = ((x_i, \rho_i), \text{sigk}_i, \rho_i^{\text{aux}})$.
- Compute $\pi_i \leftarrow \text{NIZK.Prove}(\text{crs}, \phi_i, w_i)$.
- Set $\overline{\text{msg}}_i^1 = (\text{msg}_i^1, \{ct_j^{\text{GC}_i}\}_{j \in [n]}, \{K_i^\alpha\}_{\alpha \in [|\text{inp}|]}, \{\text{sh}_i^\alpha\}_{\alpha = |\text{inp}| + 1}^{\#\text{wires}}, (\phi_i, \pi_i))$ and compute $\sigma_i^{\overline{\text{msg}}_i^1} \leftarrow \text{DS.Sign}(\text{sigk}_i, \overline{\text{msg}}_i^1)$.
- Run round 1 of $\Pi_{\text{BC-PISA}}$ with input $(\overline{\text{msg}}_i^1, \sigma_i^{\overline{\text{msg}}_i^1})$.

Round 2: Each party $\mathcal{P}_i$ does the following:

- For $j \in [n]$, let $\overline{\text{msg}}_j^1$ be the message received in round 1 of $\Pi_{\text{BC-PISA}}$ invoked by $\mathcal{P}_j$ as sender, and parse it as above.
- Let $\mathcal{S}_i^{\text{complain}} = \{\}$.
- For $j \in [n]$, check that $\text{NIZK.Verify}(\text{crs}, \phi_j, \pi_j) = 1$ and $\text{DS.Ver}(\text{vk}_j, \overline{\text{msg}}_j^1, \sigma_j^{\overline{\text{msg}}_j^1}) = \text{accept}$, and otherwise add j to $\mathcal{S}_i^{\text{complain}}$.
- For $j \notin \mathcal{S}_i^{\text{complain}}$, compute $([\text{GC}_j]_i, \sigma_i^{\text{GC}_j}) \leftarrow \text{Dec}(\text{sk}_i^{\text{GC}}, ct_j^{\text{GC}_j})$.
- For $j \in [n]$ and $\beta \in [|\text{msg}|]$, let $\alpha = |\text{inp}| + (j - 1)|\text{msg}| + \beta$. Set $v_i^\alpha = \perp$ if $j \in \mathcal{S}_i^{\text{complain}}$, and $v_i^\alpha = \text{msg}_j^1[\beta]$ otherwise.
- For $k \in [n]$ and $\alpha \in \{|\text{inp}| + 1, \dots, \#\text{wires}\}$, compute $\overline{\text{sh}}_{i \rightarrow k}^\alpha \leftarrow \text{1or0wo.vote}(v_i^\alpha, \text{sk}_i^{k, \alpha})$.
- Set $\overline{\text{msg}}_i^2 = (\{([\text{GC}_j]_i, \sigma_i^{\text{GC}_j})\}_{j \in [n] \setminus \mathcal{S}_i^{\text{complain}}}, \{\overline{\text{sh}}_{i \rightarrow k}^\alpha\}_{k \in [n], \alpha \in \{|\text{inp}| + 1, \dots, \#\text{wires}\}})$.
- Compute $\sigma_i^2 \leftarrow \text{DS.Sign}(\text{sigk}_i, \overline{\text{msg}}_i^2)$.
- For $j \in [n]$, run round 2 of the corresponding $\Pi_{\text{BC-PISA}}$ instance.
- Send $(\overline{\text{msg}}_i^2, \sigma_i^2)$ to every party.

Output: Each party $\mathcal{P}_i$ does the following:

- Let $\mathcal{S}^{\text{invalid}} = \{\}$.
- For $j \in [n]$, check $\text{DS.Ver}(\text{vk}_j, \overline{\text{msg}}_j^2, \sigma_j^2) = \text{accept}$ and add j to $\mathcal{S}^{\text{invalid}}$ if the check fails.
- For $j \notin \mathcal{S}^{\text{invalid}}$, parse $\overline{\text{msg}}_j^2$ as above.
- For $j \in [n]$, obtain $\Pi_{\text{BC-PISA}}.\text{output}$ from the corresponding instance. If it is $(\text{cheat}/\text{nr}, \pi_\perp)$, output $(\text{cheat}/\text{nr}, (\text{BC}, j, \pi_\perp))$ and stop.
- For $j \in [n]$, parse the successful $\Pi_{\text{BC-PISA}}.\text{output}$ as $\overline{\text{msg}}_j^1$ and check that $\text{NIZK.Verify}(\text{crs}, \phi_j, \pi_j) = 1$. On failure, output $(\text{cheat}, (\text{NIZK}, j, \Pi_{\text{BC-PISA}}.\text{output}))$ and stop.

- (Check the well-formedness of parties' ballots, and create a vector $\mathcal{B}$ of well-formed ballots) For $j, k \in [n]$ and $\beta \in [|\text{msg}|]$, let $\alpha = |\text{inp}| + (j - 1)|\text{msg}| + \beta$. Set $\mathcal{B}_{k,\alpha} \leftarrow (\perp)_{r \in [n]}$. For $r \in [n] \setminus \mathcal{S}^{\text{invalid}}$, set $\mathcal{B}_{k,\alpha}[r] \leftarrow \overline{\text{sh}}_{r \rightarrow k}^\alpha$ if $\overline{\text{sh}}_{r \rightarrow k}^\alpha[1] \in \{\text{msg}_j^1[\beta], \perp\}$ and $\text{KeyVer}(\text{pk}_r^{k,\alpha,\overline{\text{sh}}_{r \rightarrow k}^\alpha[1]}, \overline{\text{sh}}_{r \rightarrow k}^\alpha[2]) = 1$.
- For $k \in [n]$ and $\alpha \in \{|\text{inp}| + 1, \dots, \#\text{wires}\}$, set $\mathcal{T}_{k,\alpha} \leftarrow \{r \in [n] : \mathcal{B}_{k,\alpha}[r] \neq \perp\}$ and compute $K_k^\alpha \leftarrow \text{1or0wo.reconstruct}(\text{sh}_k^\alpha, \{\mathcal{B}_{k,\alpha}[r]\}_{r \in \mathcal{T}_{k,\alpha}})$.
- For $j \in [n]$, set $\mathcal{S}_j^{\text{gc}} \leftarrow \{k \in [n] \setminus \mathcal{S}^{\text{invalid}} : \text{DS.Ver}(\text{vk}_j, (k, [\text{GC}_j]_k), \sigma_k^{\text{gc}_j}) = \text{accept}\}$.
- For $j \in [n]$, compute $\text{GC}_j \leftarrow \text{shamir.reconstruct}_{2t+1}(\{(k, [\text{GC}_j]_k)\}_{k \in \mathcal{S}_j^{\text{gc}}})$. In case the reconstruction fails, set $\pi_\perp \leftarrow \{(k, [\text{GC}_j]_k, \sigma_k^{\text{gc}_j})\}_{k \in \mathcal{S}_j^{\text{gc}}}$ and output $(\text{cheat}, (\text{GC}, j, \pi_\perp))$, and stop.
- For $j \in [n]$, compute $\text{msg}_j^2 \leftarrow \text{Eval}(\text{GC}_j, \{K_j^\alpha\}_{\alpha \in [\#\text{wires}]})$.
- Compute $y \leftarrow \text{out}(\{\text{msg}_j^1\}_{j \in [n]}, \{\text{msg}_j^2\}_{j \in [n]})$. Output (out, y) .

Audit: $\text{audit}(\pi) \rightarrow \{\text{accept}, \text{reject}\}$:

- If $\pi = (\text{BC}, j, \pi_\perp)$, output $\Pi_{\text{BC-PISA}}.\text{audit}(\pi_\perp)$.
- If $\pi = (\text{NIZK}, j, \pi_\perp)$, parse $\pi_\perp = (\overline{\text{msg}}_j^1, \sigma_j^{\text{BC}})$ and $\overline{\text{msg}}_j^1$ as above. Output **accept** if $\text{DS.Ver}(\text{vk}_j, \overline{\text{msg}}_j^1, \sigma_j^{\text{BC}}) = \text{accept}$ and $\text{NIZK.Verify}(\text{crs}, \phi_j, \pi_j) = 0$, and **reject** otherwise.
- Otherwise, parse $\pi = (\text{GC}, j, \pi_\perp)$ and $\pi_\perp = \{(k, [\text{GC}_j]_k, \sigma_k^{\text{gc}_j})\}_{k \in \mathcal{S}_j^{\text{gc}}}$.
- If $|\mathcal{S}_j^{\text{gc}}| < 2t + 1$, output **reject** and stop.
- For $k \in \mathcal{S}_j^{\text{gc}}$, check that $\text{DS.Ver}(\text{vk}_j, (k, [\text{GC}_j]_k), \sigma_k^{\text{gc}_j}) = \text{accept}$ and output **reject** if any check fails.
- Compute $\text{GC}_j \leftarrow \text{shamir.reconstruct}_{2t+1}(\{(k, [\text{GC}_j]_k)\}_{k \in \mathcal{S}_j^{\text{gc}}})$. Output **accept** if reconstruction fails, and **reject** otherwise.

Fig. 4: 2-round PISA Protocol for $n > 4t$

6.3 Security Proof

Theorem 5. *Let $n > 4t$ and $n = O(\log(\lambda))$, and assume a setup with crs and PKI (as in Figure 4). Given a 2-round broadcast-only protocol Π_{GOD} with guaranteed output delivery (with straight-line simulation and first-round input extraction) and assuming a secure One-or-Nothing Secret Sharing with Omissions scheme, NIZK scheme, garbling scheme, key-verifiable PKE scheme, and a signature scheme, $\Pi_{\text{PISA-2}}$ realizes $\mathcal{F}_{\text{PISA}}$ over point-to-point channels in two rounds, against a malicious adversary corrupting t parties.*

Below we describe the simulator $\mathcal{S}$ for a real-world adversary $\mathcal{A}$. We use the NIZK simulator $\text{simP}(\text{crs}, \tau, \phi)$ and extractor $\text{E}(\tau, \phi, \pi)$ from the preliminaries,

with accepting simulation also for statements without witnesses and extraction for every new statement–proof pair. We use the garbling simulator simGC of Section 3.4, which takes the previously sampled active labels as input and allows the input to be chosen after these labels are sampled. Encryptions of 0 have the same plaintext length as the encryptions they replace.

Simulator $\mathcal{S}$

Let $\mathcal{H}$ and I denote the sets of honest and corrupt parties, respectively, and let $c = |I|$. Let $\mathcal{S}_{\text{GOD}}$ denote the simulator for the underlying protocol Π_{GOD} , and let $\mathcal{F}_{\text{GOD}}$ and $\mathcal{F}_{\text{PISA}}$ denote the ideal functionalities corresponding to Π_{GOD} and Π_{PISA} , respectively. Start by invoking $\mathcal{S}_{\text{GOD}}$ for the underlying setup, generate the honest parties' remaining keys as in the protocol, and compute $(\text{crs}, \tau) \leftarrow \text{Gen}(1^\lambda, \mathcal{R})$. Receive messages $\{\text{msg}_i^1\}_{i \in \mathcal{H}}$ from $\mathcal{S}_{\text{GOD}}$. Proceed with the following steps:

Round 1: On behalf of each honest party $\mathcal{P}_i \in \mathcal{H}$ do the following:

- Sample one label K_i^α for every wire $\alpha \in [\#\text{wires}]$ (as in KeyGen).
- For $\alpha \in \{|\text{inp}| + 1, \dots, \#\text{wires}\}$, compute $\text{sh}_i^\alpha \leftarrow \text{1or0wo.share}((K_i^\alpha, K_i^\alpha), \{\text{pk}_j^{i,\alpha}\}_{j \in [n]})$.
- Compute $\{[0]_j\}_{j \in [n]} \leftarrow \text{shamir.share}(0^{|\mathcal{C}_i|}, n, 2t + 1)$ and retain $[\text{GC}_i]_j = [0]_j$ for $j \in I$.
- For $j \in I$, sign and encrypt $[\text{GC}_i]_j$ as in the protocol. For $j \in \mathcal{H}$, set $\text{ct}_j^{\text{GC}_i} \leftarrow \text{Enc}(\text{pk}_j^{\text{GC}}, 0)$.
- Set ϕ_i as in the protocol and compute $\pi_i \leftarrow \text{simP}(\text{crs}, \tau, \phi_i)$.
- Complete the remainder of the first round as in the protocol.

Input Extraction: Upon completing the above steps, obtain first-round messages from $\mathcal{A}$ to each honest party. For every authenticated corrupt first-round bundle with $\text{NIZK.Verify}(\text{crs}, \phi, \pi) = \text{accept}$, compute $w \leftarrow \text{E}(\tau, \phi, \pi)$ and stop the simulation if $(\phi, w) \notin \mathcal{R}$. To obtain the first-round message of each corrupt party $\mathcal{P}_j \in I$, do the following steps:

- Compute the honest parties' complaint sets and votes as in the protocol.
- For $\beta \in [|\text{msg}|]$, let $\alpha = |\text{inp}| + (j - 1)|\text{msg}| + \beta$ and set $\mathcal{S}_b^\alpha = \{i \in \mathcal{H} : v_i^\alpha = b\}$ for $b \in \{0, 1, \perp\}$.
- For $\beta \in [|\text{msg}|]$, let $\alpha = |\text{inp}| + (j - 1)|\text{msg}| + \beta$ and set $\widehat{\text{msg}}_j^1[\beta] = b$ if $c + |\mathcal{S}_b^\alpha| \geq n - 2t$ and $c + |\mathcal{S}_b^\alpha| + |\mathcal{S}_\perp^\alpha| \geq n - t$ for $b \in \{0, 1\}$; otherwise set $\widehat{\text{msg}}_j^1[\beta] = 0$.

The bit b is unique whenever it exists, since $n > 4t$. For $j \in I$, set $\widehat{\text{msg}}_j^1 = \widehat{\text{msg}}_j^1[1] \parallel \dots \parallel \widehat{\text{msg}}_j^1[|\text{msg}|]$ and send $\{\widehat{\text{msg}}_j^1\}_{j \in I}$ to $\mathcal{S}_{\text{GOD}}$. Upon receiving a request from $\mathcal{S}_{\text{GOD}}$ to query $\mathcal{F}_{\text{GOD}}$ with inputs $\{x_j\}_{j \in I}$, forward this message to $\mathcal{F}_{\text{PISA}}$ and receive output y . Forward y to $\mathcal{S}_{\text{GOD}}$ and receive $\{\text{msg}_i^2\}_{i \in \mathcal{H}}$ as a response.

Round 2: For each $\mathcal{P}_i \in \mathcal{H}$ do the following:

- Compute $\text{GC}_i \leftarrow \text{simGC}(C_i, \text{msg}_i^2, \{K_i^\alpha\}_{\alpha \in [\#\text{wires}]})$.
- Sample a Shamir sharing of GC_i with threshold $2t + 1$, uniformly conditioned on $\{[\text{GC}_i]_j\}_{j \in I}$ fixed in round 1.
- For $j \in \mathcal{H}$, compute $\sigma_j^{\text{GC}_i} \leftarrow \text{DS.Sign}(\text{sigk}_i, (j, [\text{GC}_i]_j))$.

For each $\mathcal{P}_i \in \mathcal{H}$, complete the remainder of the second round as in the protocol, using the above shares and signatures in place of the decrypted GC shares of honest garblers.

Output:

- For every authenticated corrupt first-round bundle first revealed in round 2, if $\text{NIZK.Verify}(\text{crs}, \phi, \pi) = \text{accept}$, compute $w \leftarrow \text{E}(\tau, \phi, \pi)$. If $(\phi, w) \notin \mathcal{R}$, stop the simulation.
- For each $\mathcal{P}_i \in \mathcal{H}$, complete the output phase as in the protocol.
- For each $\mathcal{P}_i \in \mathcal{H}$ whose output is (out, y_i) , set $\text{out}_i = y$. If $y_i \neq y$, stop the simulation.
- For each $\mathcal{P}_i \in \mathcal{H}$ whose output is $(\text{cheat}, \pi_{\perp i})$, let (type_i, j) be the cheating type and party identified by $\pi_{\perp i}$. If $\text{audit}(\pi_{\perp i}) = \text{reject}$ or $j \in \mathcal{H}$, stop the simulation. Otherwise, set $\text{out}_i = (\text{type}_i, j)$ and retain $\pi_{\perp i}$.
- Send $\{\text{out}_i\}_{i \in \mathcal{H}}$ to the trusted party as the honest parties' selective outputs.
- If any honest party obtains any other output, stop the simulation.
- Output the output of $\mathcal{A}$ and terminate.

Throughout the simulation, stop if $\mathcal{A}$ produces a valid signature under an honest party's verification key on a message not signed on behalf of that party. The cheating type is `nr` for BC non-responsiveness certificates and `cheat` for the remaining certificates.

Fig. 5: Simulator

Lemma 2 (Output validity lemma). *Let P be an honest party that outputs neither a BC nor a NIZK certificate, and let P_j be corrupt. Then:*

1. *the message msg_j^1 that P obtains from $\Pi_{\text{BC-PISA}}^j$ equals $\widehat{\text{msg}}_j^1$;*
2. *for every garbler P_k and every $\beta \in [|\text{msg}|]$, the honest parties' ballots for wire $\alpha = |\text{inp}| + (j - 1)|\text{msg}| + \beta$ of P_k 's circuit contain at least $n - 2t$ votes for $\text{msg}_j^1[\beta]$, and all other honest ballots vote $\perp$;*
3. *P receives validly signed shares of GC_j from at least $n - 2t \geq 2t + 1$ honest parties.*

Proof (sketch). Every honest party $\mathcal{P}_i$ that received a validly signed first-round message from $\mathcal{P}_j$ received msg_j^1 , since otherwise its echo would give $\mathcal{P}_i$ two conflicting signed messages. At most t honest parties received no message, since

otherwise $\mathcal{P}_i$ would obtain more than t signed complaints of non-receipt. Moreover, $\mathcal{P}_j$'s proof verifies, as otherwise $\mathcal{P}_i$ would output a NIZK certificate. Hence at least $|\mathcal{H}| - t \geq n - 2t$ honest parties vote $\text{msg}_j^1[\beta]$ and decrypt and forward their shares of GC_j , while all other honest parties vote $\perp$. This gives items 2 and 3, where $n - 2t \geq 2t + 1$ uses $n > 4t$. For item 1, it gives $c + |S_b^\alpha| \geq n - 2t$ and $c + |S_b^\alpha| + |S_\perp^\alpha| = n \geq n - t$ for $b = \text{msg}_j^1[\beta]$, so the simulator sets $\widehat{\text{msg}}_j^1[\beta] = b$. This bit is unique since $n > 4t$.

The same conclusions hold for an honest sender, with every honest party receiving its first-round message. Thus, for every sender, the honest ballots alone satisfy both reconstruction thresholds. Accepted additional ballots cannot change the reconstructed labels, by key verifiability and correctness of **1or0wo**. For a corrupt garbler, the extracted witness fixes a consistent GC sharing. If the received signed shares are consistent, the $2t + 1$ correct shares fix the reconstructed circuit; otherwise, they give the GC certificate.

We introduce a series of hybrid experiments to demonstrate that the joint distribution of the adversary's output and the outputs of the honest parties in the ideal execution is computationally indistinguishable from the corresponding joint distribution obtained in a real protocol execution. In each experiment, the output is defined as the tuple consisting of the adversary's output and the outputs of the honest parties. We represent a cheating output by its cheating type and identified party; the corresponding certificate is retained for the public audit. Whenever a first-round message changes, its statement, simulated proof, and authentication are generated for the changed message.

Hybrid H_0 (Real World): Execute the protocol as in a real execution, using the honest parties' inputs.

Hybrid H_1 (Signature Unforgeability): Same as the previous hybrid, except that the simulator stops if $\mathcal{A}$ produces a valid signature under an honest party's verification key on a message not signed on behalf of that party. Note that now no certificates of cheating can be produced against the honest parties

This hybrid is indistinguishable from the previous one by the unforgeability of the signature scheme.

Hybrid H_2 (Trapdoored CRS): Same as the previous hybrid, except that the **crs** is generated as a part of a pair $(\text{crs}, \tau) \leftarrow \text{Gen}(1^\lambda, \mathcal{R})$ and the trapdoor τ is retained.

This hybrid is identically distributed to the previous one, since the distribution of the reference string is unchanged.

Hybrid H_3 (Simulated Proofs): Same as the previous hybrid, except that for each honest party $\mathcal{P}_i$, the proof for relation $\mathcal{R}$ is generated using $\pi_i \leftarrow \text{simP}(\text{crs}, \tau, \phi_i)$.

This hybrid is indistinguishable from the previous one by the zero knowledge of the NIZK scheme.

Hybrid H_4 (Witness Extraction): Same as the previous hybrid, except that for every message with a verifying signature, whenever $\text{NIZK.Verify}(\text{crs}, \phi, \pi) =$

accept, the simulator extracts a witness as $w \leftarrow \mathbf{E}(\tau, \phi, \pi)$ and stops the simulation if $(\phi, w) \notin \mathcal{R}$. This applies also to messages first revealed in round 2.

This hybrid is indistinguishable from the previous one by the simulation extractability of the NIZK scheme.

Hybrid H_5 (1or0wo Correctness): Same as the previous hybrid, except that the simulator stops if the honest parties ever get inconsistent labels as the output of the 1or0wo.

This hybrid is indistinguishable from the previous one by the correctness of the 1or0wo.

Hybrid H_6 (Correct Output): Same as the previous hybrid, except that the simulator sends the extracted input to the ideal functionality and receives ideal output. If there exists an honest party that doesn't abort but receives output that differs from the ideal output, abort the simulation.

This hybrid is indistinguishable from the previous one by Lemma 2, which shows that the additional stopping condition occurs only with negligible probability.

Hybrid H_7 (Correct Accusations): Same as the previous hybrid, except that the simulator sends each honest party's output to the ideal functionality. By the previous hybrid, if it's not an abort, then the output is correct. If it is an abort, then it is a valid accusation of a corrupt party.

This hybrid is indistinguishable from the previous one, as all outputs seen by the distinguisher are the same.

Hybrid H_8 (Encrypt 0s): Same as the previous hybrid, except that each honest party encrypts all other honest parties' shares as an encryption of 0: $ct_j^{\text{GC}} \leftarrow \text{Enc}(\text{pk}_j^{\text{GC}}, 0)$. Honest parties use their original shares and signatures in round 2.

This hybrid is indistinguishable from the previous one by the CPA security of the encryption scheme. The corresponding decryption keys are never revealed.

Hybrid H_9 (Delayed Garbling): Same as the previous hybrid, except that for each honest party $\mathcal{P}_i$, the garbled circuit is generated after round 1. The labels are generated in round 1 as before, and the GC shares for corrupt recipients are sampled as shares of $0^{|C_i|}$. After generating GC_i , sample its Shamir sharing uniformly conditioned on these shares, and sign the honest recipients' shares.

This hybrid is identically distributed to the previous one by independent label generation and the privacy of Shamir secret sharing: at most $c \leq t$ shares are fixed before the circuit is generated.

Hybrid H_{10} (Active Labels): Same as the previous hybrid, except that for each honest party $\mathcal{P}_i$, the pair of labels used in a given execution of 1or0wo are identical. We let this label be the "active" label for the message bit to which this execution of 1or0wo corresponds.

In honest parties' garbled circuits, two different labels are still used (the circuit is garbled honestly, as before). Corrupt party messages $\{\widehat{\text{msg}}_j^1\}_{j \in I}$ are obtained from the honest parties' votes as in the simulator, and $\widehat{\text{msg}}_j^1 = \text{msg}_j^1$ for $j \in \mathcal{H}$. Honest parties' circuits are garbled so as to place the "active" label

in the position corresponding to these messages; the other label is sampled as prescribed by the garbling scheme. The labels for the parties' own inputs and randomness are unchanged.

This hybrid is indistinguishable from the previous one by the privacy of 1or0wo.

Hybrid H_{11} (Simulated Garbled Circuit): Same as the previous hybrid, except that for each honest party $\mathcal{P}_i$, the garbled circuit is generated using $GC_i \leftarrow \text{simGC}(C_i, \text{msg}_i^2, \{K_i^\alpha\}_{\alpha \in [\#\text{wires}]})$, where $\text{msg}_i^2 \leftarrow \text{snd-msg}_i(x_i, \rho_i, \{\widehat{\text{msg}}_j\}_{j \in [n]})$. The active labels are preserved, and the sharing is completed as before.

This hybrid is indistinguishable from the previous one by the privacy of the garbling scheme.

Hybrid H_{12} (Simulated Messages): Same as the previous hybrid, except that the underlying setup and the honest parties' first- and second-round messages are generated using $\mathcal{S}_{\text{GOD}}$.

This hybrid is indistinguishable from the previous one by the security and first-round input extraction of Π_{GOD} .

Hybrid H_{13} (Ideal World): Same as the previous hybrid, except that the distinguisher gets honest parties' outputs from the ideal functionality (rather than from our simulation). *This hybrid is indistinguishable from the previous one because we already argued that the outputs are correct and informed the ideal functionality what accusations to make (hybrid 7).*

In Hybrid H_{13} , the simulator no longer depends on any hidden information of the functionality, and honest parties output the correct output or identify a corrupt party with an accepted certificate, as per the definition of PISA. Different honest parties may obtain different selective outputs. Therefore, the joint output distribution in the last hybrid is identical to that in the ideal-world experiment, completing the simulation argument.

References

- ACGJ18. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 395–424, August 2018.
- BDO14. Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14: 9th International Conference on Security in Communication Networks*, volume 8642 of *Lecture Notes in Computer Science*, pages 175–196, September 2014.
- BL18. Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532, April / May 2018.

- BMRS24. Carsten Baum, Nikolas Melissaris, Rahul Rachuri, and Peter Scholl. Cheater identification on a budget: MPC with identifiable abort from pairwise MACs. *Lecture Notes in Computer Science*, pages 454–488, August 2024.
- BOSS20. Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 562–592, August 2020.
- CDKs24. Ran Cohen, Jack Doerner, Yashvanth Kondi, and abhi shelat. Secure multiparty computation with identifiable abort via vindicating release. *Lecture Notes in Computer Science*, pages 36–73, August 2024.
- CDvdG88. David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party’s input and correctness of the result. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO’87*, volume 293 of *Lecture Notes in Computer Science*, pages 87–119, August 1988.
- CGZ20. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 828–858, May 2020.
- CL17. Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, October 2017.
- DMR⁺21. Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakubov. Broadcast-optimal two round MPC with an honest majority. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 155–184, Virtual Event, August 2021.
- DRSY23a. Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakubov. Broadcast-optimal two round MPC with asynchronous peer-to-peer channels. In *LATINCRYPT*, volume 14168 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 2023.
- DRSY23b. Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakubov. Minimizing setup in broadcast-optimal two round MPC. In *Advances in Cryptology – EUROCRYPT 2023, Part II*, *Lecture Notes in Computer Science*, pages 129–158, June 2023.
- GGHR14. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94, February 2014.
- GIKR01. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd Annual ACM Symposium on Theory of Computing*, pages 580–589. ACM Press, July 2001.
- GIKR02. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 178–193, August 2002.

- GJPR21. Aarushi Goel, Abhishek Jain, Manoj Prabhakaran, and Rajeev Raghunath. On communication models and best-achievable security in two-round MPC. In Kobbi Nissim and Brent Waters, editors, *TCC 2021: 19th Theory of Cryptography Conference, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 97–128, November 2021.
- GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 63–82, August 2015.
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229. ACM Press, May 1987.
- GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499, April / May 2018.
- IOZ14. Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 369–386, August 2014.
- KR25. Yashvanth Kondi and Divya Ravi. Separating broadcast from cheater identification. In *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, pages 3664–3677. ACM Press, November 2025.
- KRSY26. Yashvanth Kondi, Divya Ravi, Jure Sternad, and Sophia Yakoubov. Putting the OPTI in round optimal IA-MPC in the plain model. Cryptology ePrint Archive, Paper 2026/406, 2026. To appear in ASIACRYPT 2026.
- MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763, May 2016.
- PR18. Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 425–458, August 2018.
- Riv25. Marc Rivinius. MPC with publicly identifiable abort from pseudorandomness and homomorphic encryption. *Lecture Notes in Computer Science*, pages 270–300, June 2025.
- Sha79. Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.
- Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, October 1986.