Broadcast-Optimal Two Round MPC
with an Honest Majority

Ivan Damgård[1], Bernardo Magri[1,2], Divya Ravi[1], Luisa Siniscalchi[1,2], and
Sophia Yakoubov[1]*

[1] Aarhus University, Denmark
[2] Concordium Blockchain Research Center

Abstract. This paper closes the question of the possibility of two-round
MPC protocols achieving different security guarantees with and without
the availability of broadcast in any given round. Cohen *et al.* [CGZ20]
study this question in the dishonest majority setting; we complete the
picture by studying the honest majority setting.

In the honest majority setting, given broadcast in both rounds, it is
known that the strongest guarantee — guaranteed output delivery —
is achievable [GLS15]. We show that, given broadcast in the first round
only, guaranteed output delivery is still achievable. Given broadcast in
the second round only, we give a new construction that achieves identi-
fiable abort, and we show that fairness — and thus guaranteed output
delivery — are not achievable in this setting. Finally, if only peer-to-peer
channels are available, we show that the weakest guarantee — selective
abort — is the only one achievable for corruption thresholds $t > 1$ and
for $t = 1$ and $n = 3$. On the other hand, it is already known that selective
abort can be achieved in these cases. In the remaining cases, i.e., $t = 1$
and $n \geq 4$, it is known [IKP10,IKKP15] that guaranteed output delivery
(and thus all weaker guarantees) are possible.

1 Introduction

In this paper we advance the study of round-optimal secure computation, focus-
ing on secure computation with active corruptions, an honest majority, and some
setup (e.g. a public key infrastructure). It is known that in this setting, secure
computation is possible in two rounds (whereas one round is clearly not enough).
However, most known two-round protocols in the honest majority setting either
only achieve the weakest security guarantee (selective abort) [ACGJ19], or make
use of a broadcast channel in both rounds [GLS15]. Since broadcast channels are
expensive, it is important to try to minimize their use (while achieving strong
security guarantees).

The only step in this direction is the protocol of Cohen *et al.* [CGZ20]. They
achieve secure computation with unanimous abort for a dishonest majority (and

---

thus also for an honest majority) with broadcast in the second round only, and they also show that unanimous abort is the strongest achievable guarantee in this setting. Finally, Cohen *et al.* showed that, given a dishonest majority, selective abort is the strongest achievable security guarantee with broadcast in the first round only.

We make a study analogous to the work of Cohen *et al.* but in the honest majority setting. Like Cohen *et al.*, we consider all four broadcast patterns: broadcast in both rounds, broadcast in the second round only, broadcast in the first round only, and no broadcast at all. Gordon *et al.* [GLS15] showed that, given broadcast in both rounds, the strongest guarantee — guaranteed output delivery — is achievable. For each of the other broadcast patterns, we ask:

> What is the strongest achievable security guarantee in this broadcast pattern, given an honest majority?

We consider the following security guarantees:

Selective Abort (SA): A secure computation protocol achieves *selective abort* if every honest party either obtains the output, or aborts.
Unanimous Abort (UA): A secure computation protocol achieves *unanimous abort* if either *all* honest parties obtain the output, or they all (unanimously) abort.
Identifiable Abort (IA): A secure computation protocol achieves *identifiable abort* if either all honest parties obtain the output, or they all (unanimously) abort, *identifying one corrupt party.*
Fairness (FAIR): A secure computation protocol achieves *fairness* if either all parties obtain the output, or none of them do. In particular, an adversary cannot learn the output if the honest parties do not also learn it.
Guaranteed Output Delivery (GOD): A secure computation protocol achieves *guaranteed output delivery* if all honest parties will learn the computation output no matter what the adversary does.

Some of these guarantees are strictly stronger than others. In particular, guaranteed output delivery implies identifiable abort (since an abort never happens), which implies unanimous abort, which in turn implies selective abort. Similarly, guaranteed output delivery implies fairness, which implies unanimous abort. Fairness and identifiable abort are incomparable. In a fair protocol, in case of an abort, both corrupt and honest parties get less information: corrupt parties are guaranteed to learn nothing if the protocol aborts, but honest parties may not learn anything about corrupt parties' identities. On the other hand, in a protocol with identifiable abort, in case of an abort corrupt parties may learn the output, but honest parties will identify at least one corrupt party.

In Table 1, we summarize our results. Like the impossibility results of Cohen *et al.*, all of our impossibility results hold given arbitrary setup (such as a common reference string, a public key infrastructure, and correlated randomness). Our feasibility results use only a PKI and CRS. Below we give a very brief description of our results. It turns out that going from dishonest to honest majority

Broadcast Pattern		t	selective abort	unanimous abort	identifiable abort	fairness	guaranteed output delivery
R1	R2						
BC	BC		✓ ←—	— ✓ ←—	✓ [GLS15]	✓ ←—	— ✓ [GLS15]
P2P	BC	$1 < t < \frac{n}{2}$	✓ ←—	— ✓ ←—	✓ (Thm 9)	✗ (Thm 2) for $t > 1$ ✗ (Cor 3) for $n \leq 3t$	↛ ✗ for $t > 1$ ↛ ✗ for $n \leq 3t$
BC	P2P		✓ ←—	— ✓ ←—	✓ (Thm 7)	✓ ←—	— ✓ (Thm 7)
P2P	P2P		✓ [ACGJ19]	✗ (Cor 1) for $t > 1$	↛ ✗ for $t > 1$	✗ (Thm 2) for $t > 1$ ✗ (Cor 3) for $n \leq 3t$	↛ ✗ for $t > 1$ ↛ ✗ for $n \leq 3t$
P2P	P2P	$t = 1,\ n = 3$	✓ [ACGJ19]	✗ (Cor 2) ——	—→ ✗	✗ (Cor 2) ——	—→ ✗
		$t = 1,\ n = 4$	✓ ←—	— ✓ ←—	✓ ([IKKP15])	✓ ←—	— ✓ ([IKKP15])
		$t = 1,\ n \geq 5$	✓ ←—	— ✓ ←—	✓ ([IKP10])	✓ ←—	— ✓ ([IKP10])

Table 1: Feasibility and impossibility for two-round MPC in the honest majority setting with different guarantees and broadcast patterns.
The R1 column describes whether broadcast is available in round 1; the R2 column describes whether broadcast is available in round 2.
Arrows indicate implication: the possibility of a stronger security guarantee implies the possibility of weaker ones in the same setting, and the impossibility of a weaker guarantee implies the impossibility of stronger ones in the same setting.

allows for stronger security guarantees in some, but not all cases. In section 1.1 we give a longer overview of our results, and the techniques we use.

No Broadcast In this setting, we show that if the adversary controls two or more parties ($t > 1$), or if $t = 1, n = 3$, selective abort is the best achievable guarantee. This completes the picture, since (1) selective abort can indeed be achieved by the results of Ananth *et al.* [ACGJ19], and (2) for $t = 1, n \geq 4$, guaranteed output delivery can be achieved by the results of Ishai *et al.* [IKP10], [IKKP15].

Broadcast in the First Round Only In this setting, we show that guaranteed output delivery — the strongest guarantee — can be achieved.

Broadcast in the Second Round Only In this setting, we show that fairness is impossible if $t \geq n/3$, or if $t > 1$ (again, in the remaining case of $t = 1, n \geq 4$, guaranteed output delivery can be achieved). If fairness is ruled out, the best one can hope for is identifiable abort, and we show this can indeed be achieved given an honest majority.

To achieve identifiable abort with broadcast in the second round only, we introduce a new tool called *one-or-nothing secret sharing*, which we believe to be

of independent interest. One-or-nothing secret sharing is a flavor of secret sharing that allows a dealer to share a *vector* of secrets. Once the shares are distributed to the receivers, they can vote on which secret to reconstruct by publishing "ballots". Each receiver either votes for the secret she wishes to reconstruct, or abstains (by publishing a special equivocation ballot). If only one secret is voted for, and gets sufficiently many votes, the ballots enable reconstruction of that secret. On the other hand, if receivers disagree about which secret to reconstruct, nothing is revealed. This could have applications to voting scenarios where, though some voters may remain undecided, unanimity among the decided voters is important.

1.1 Technical Overview

In this section we summarize our results given each of the broadcast patterns in more detail.

No Broadcast (P2P-P2P) Without a broadcast channel, we show that only the weakest guarantee — selective abort — is achievable. Ananth *et al.* [ACGJ19] give a protocol for secure computation with selective abort in this setting; we prove that secure computation with unanimous abort is not achievable, implying impossibility for all stronger guarantees. More specifically, we get the following two results:

Result 1 (Cor 1: P2P-P2P, UA, $t > 1$) *Secure computation of general functions with unanimous abort cannot be achieved in two rounds of peer-to-peer communication for corruption threshold $t > 1$.*

Result 2 (Cor 2: P2P-P2P, UA, $t = 1$, $n = 3$) *Secure computation of general functions with unanimous abort cannot be achieved in two rounds of peer-to-peer communication for corruption threshold $t = 1$ when $n = 3$ [3].*

We prove the first result by focusing on broadcast, where only one party (the dealer) has an input bit, and all parties should output that bit. We show that computing broadcast with unanimous abort in two peer-to-peer rounds with $t > 1$ is impossible[4].

The only case not covered by these two results is $t = 1$ and $n \geq 4$. However for this case, it follows from results by Ishai *et al.* [IKP10] and [IKKP15] that the strongest guarantee — guaranteed output delivery — is achievable in two rounds of peer-to-peer communication.

For completeness, we note that the case of $n = 2$ and $t = 1$ is special. We are no longer in an honest majority setting, so fairness is known to be impossible [Cle86]. The other three guarantees are possible and equivalent.

---

[3] Patra and Ravi [PR18] give a similar result in the absence of a PKI and correlated randomness; our impossibility result is stronger, as it holds even given arbitrary correlated randomness

[4] It is well known that computing broadcast with guaranteed output delivery requires t rounds, but this of course does not imply the same for broadcast with unanimous abort.

Broadcast in the First Round Only (BC-P2P) We show that any *first-round extractable* two broadcast-round protocol (where the simulator demonstrating security of the protocol can extract parties' inputs from their first-round messages and it is efficient to check whether a given second-round message is correct) can be run over one broadcast round followed by one peer-to-peer round without any loss in security. Since the protocol of Gordon *et al.* [GLS15] satisfies these properties, we conclude that guaranteed output delivery is achievable in the honest majority setting as long as broadcast is available in the first round.

Result 3 (Thm 7: BC-P2P, GOD, $n \geq 2t + 1$) *Secure computation of general functions with guaranteed output delivery is possible in two rounds of communication, only the first of which is over a broadcast channel, for corruption threshold t such that $n \geq 2t + 1$.*

Broadcast in the Second Round Only (P2P-BC) When broadcast is available in the second round, not the first, it turns out that fairness (and hence guaranteed output delivery) cannot be achieved. More specifically, we obtain the following two results:

Result 4 (Cor 3: P2P-BC, FAIR, $n \leq 3t$) *Secure computation of general functions with fairness cannot be achieved in two rounds of communication, only the second of which is over a broadcast channel, for corruption threshold t such that $n \leq 3t$.*

Result 5 (Thm 2: P2P-BC, FAIR, $t > 1$) *Secure computation of general functions with fairness cannot be achieved in two rounds of communication, only the second of which is over a broadcast channel, for corruption threshold $t > 1$.*

Both these results are shown using the same basic idea, namely if the protocol is fair, we construct an attack in which corrupt players send inconsistent messages in the first round and then use the second round messages to obtain two different outputs, corresponding to different choices of their own input — which, of course, violates privacy.

Combining the two results, we see that fairness is unachievable when broadcast is only available in the second round (the only case not covered is $t = 1, n \geq 4$ where guaranteed output delivery is possible, as discussed above). We therefore turn to the next-best guarantee, which is identifiable abort; in Section 8, we show how to achieve it for $n > 2t$.

Result 6 (Thm 9: P2P-BC, ID, $n > 2t$) *Secure computation of general functions with identifiable abort is achievable in two rounds of communication, only the second of which is over a broadcast channel, for corruption threshold t such that $n > 2t$.*

To show this result, we use a high-level strategy adopted from Cohen *et al.* Namely, we start from any protocol that achieves identifiable abort for honest majority given two rounds of broadcast, and compile this into a protocol that

works when the first round is limited to peer-to-peer channels. While Cohen *et al.* achieve unanimous abort this way, we aim for the stronger guarantee of identifiable abort, since we assume honest majority.

To explain our technical contribution, let us follow the approach of Cohen *et al.* and see where we get stuck. The idea is to have each party broadcast a garbled circuit in the second round. This garbled circuit corresponds to the code they would use to compute their second-round message in the underlying protocol (given their input and all the first-round messages they receive). In the first round (over peer-to-peer channels), the parties additively secret share all the labels for their garbled circuit, and send their first-round message from the underlying protocol to each of their peers. In the second round (over broadcast), for each bit of first-round message she receives, each party forwards her share of the corresponding label in everyone else's garbled circuit. Cohen *et al.* used this approach to achieve unanimous abort for dishonest majority.

However, even assuming honest majority, this will not be sufficient for identifiable abort. The main issue is that corrupt parties may send inconsistent messages in the first round. This problem cannot be solved just by requiring each party to sign their first-round messages, because P_i may send an invalid signature — or nothing at all — to P_j. P_j then cannot do what she was supposed to in the second round; so, all she can do is to complain, but she cannot demonstrate any proof that P_i cheated. All honest parties now agree that either P_i or P_j is corrupt, but there is no way to tell which one. This is not an issue if we aim for unanimous abort; however, if we aim for identifiable abort, we must either find out who to blame or compute the correct output anyway, without any further interaction.

We solve this problem by introducing a new primitive we call *one-or-nothing* secret sharing. This special kind secret sharing allows a dealer to share several values simultaneously. (In our case, the values would be two garbled circuit labels for a given bit b.) The share recipients can then "vote" on which of the values to reconstruct; if they aren't sure (in our case, they wouldn't be sure if they didn't get b in the first round), they are able to "abstain", which essentially means casting their vote with the majority. As long as there are no contradictory votes and a minority of abstain votes, reconstruction of the appropriate value succeeds; otherwise, the privacy of all values is guaranteed.

We use this primitive to share the labels for the garbled circuits as sketched above. If all reconstructions succeed, we get the correct output. Otherwise, we can identify a corrupt player. By requiring parties to sign their first-round messages, we can ensure that if there are contradicting votes, all parties can agree that some party P_i sent inconsistent messages in the first round. If there is a majority of abstains, this proves that some particular P_i sent an invalid first-round message to at least one honest party.

1.2   Related Work

The quest for optimal round-complexity for secure computation protocols is a well-established topic in cryptography. Starting with the first feasibility results

from almost 35 years ago [Yao86,GMW87,BGW88,CCD88] a lot of progress has
been made in improving the round complexity of protocols [GIKR01,Lin01,CD01]
[IK02,IKP10,IKKP15,GLS15,PR18,ACGJ18,CGZ20]. In this section we detail
the prior work that specifically targets the two-round setting. We divide the
discussion into two: impossibility and feasibility results.

Result	n	t	Guarantee	CRS?	PKI?	CR?	R1	R2
[GIKR02]	any	$t \geq 2$	fairness	✓	✗	✗	BC + P2P	BC + P2P
[GLS15]	$n = 3$	$t = 1$	fairness	✓	✗	✗	BC	BC
[PR18]	$n = 3$	$t = 1$	fairness	✓	✗	✗	BC + P2P	BC + P2P
[PR18]	$n = 3$	$t = 1$	UA	✓	✗	✗	P2P	P2P
[CGZ20]	$n = 3$	$t = 2$	UA	✓	✓	✓	BC	P2P
[CGZ20]	$n = 3$	$t = 2$	IA	✓	✓	✓	P2P	BC

Table 2: Previous impossibility results. Each row in this table describes a setting
where MPC is known to be *impossible*. "UA" stand for unanimous abort, and "IA" for
identifiable abort.

Impossibility Results. Table 2 summarizes the known lower bounds on two-round
secure computation. Gennaro *et al.* [GIKR02] shed light on the optimal round-
complexity for general MPC protocols achieving fairness without correlated ran-
domness (e.g., PKI). Their model allows for communication over both authenti-
cated point-to-point channels and a broadcast channel. They show that in this
setting, three rounds are necessary for a protocol with at least $t \geq 2$ corrupt par-
ties by focusing on the computation of exclusive-or and conjunction functions. In
a slightly different model, where the parties can communicate only over a broad-
cast channel, Gordon *et al.* [GLS15] show that the existence of a fair two-round
MPC protocol for an honest majority would imply a virtual black-box program
obfuscation scheme, which would contradict the well-known impossibility result
of Barak *et al.* [BGI+01].

Patra and Ravi [PR18] investigate the three party setting. They show that
three rounds are necessary for generic secure computation achieving unanimous
abort when parties *do not* have access to a broadcast channel, and that the same
three are necessary for fairness even when parties do have a broadcast channel.
Badrinarayanan *et al.* [BMMR21] study broadcast-optimal three-round MPC
with guaranteed output delivery given an honest majority and CRS, and show
that use of broadcast in the first two rounds is necessary.

It is well known that in the dishonest majority setting fairness cannot be
achieved for generic computation [Cle86]. Cohen *et al.* [CGZ20] study the fea-
sibility of two round secure computation with unanimous and identifiable abort
in the dishonest majority setting. Their results show that considering arbitrary
setup (e.g., a PKI) and communication over point-to-point channels, achieving

unanimous abort in two rounds is not possible even if the parties are additionally allowed to communicate over a broadcast channel only in the first round, and achieving identifiable abort in two rounds is not possible even if the parties are additionally allowed to communicate over a broadcast channel only in the second round.

Result	n	t	Guarantee	PKI?	CRS?	1st round	2nd round	Assumptions
[IKP10]	$n \geq 5$	$t = 1$	GOD	✗	✗	P2P	P2P	PRG
[IKKP15]	$n = 3$	$t = 1$	SA	✗	✗	P2P	P2P	PRG
[IKKP15]	$n = 4$	$t = 1$	GOD	✗	✗	P2P	P2P	injective OWF
[GLS15]	any	$t < \frac{n}{2}$	M-GOD	✓	✓	BC + P2P	BC + P2P	dFHE
[PR18]	$n = 3$	$t = 1$	UA	✗	✗	BC + P2P	BC + P2P	GC, NICOM, eNICOM, PRG
[ACGJ18]	any	$t < \frac{n}{2}$	UA	✗	✗	BC + P2P	BC + P2P	OWF
[ACGJ18]	any	$t < \frac{n}{2}$	FS-GOD	✓	✗	BC + P2P	BC + P2P	OWF
[ACGJ18]	any	$t < \frac{n}{2}$	FS-GOD	✗	✗	BC + P2P	BC + P2P	OWF, SH-OT
[ACGJ18]	any	$t < \frac{n}{2}$	FS-GOD / SM-GOD	✓	✗	BC	BC	OWF
[GS18]	any	$t < n$	UA	✗	✓	BC	BC	2-round OT
[CGZ20]	any	$t < n$	SA	✗	✓	P2P	P2P	2-round OT
[CGZ20]	any	$t < n$	UA	✗	✓	P2P	BC	2-round OT
[CGZ20]	any	$t < n$	IA	✗	✓	BC	BC	2-round OT

Table 3: Protocols for secure MPC with two-rounds. "UA" stands for unanimous abort, "FS-GOD" for guaranteed output delivery against fail-stop adversaries, "SM-GOD" for guaranteed output delivery against semi-malicious adversaries, and "M-GOD" for guaranteed output delivery against malicious adversaries.

Feasibility Results. Table 3 summarizes known two-round secure computation constructions. While three rounds are necessary for fair MPC [GIKR02] for $t \geq 2$ (without correlated randomness), Ishai *et al.* [IKP10] show that it is possible to build generic two-round MPC with guaranteed output delivery when only a *single* party is corrupt ($t = 1$) for $n \geq 5$. Later, [IKKP15] showed the same for $n = 4$, and that selective abort is also possible for $n = 3$.

The work of [GLS15] gives a three round generic MPC protocol that guarantees output delivery and is secure against a minority of semi-honest fail-stop adversaries where parties only communicate over point-to-point channels; the same protocol can be upgraded to be secure against malicious adversaries if the parties are also allowed to communicate over a broadcast channel. The use of broadcast channel in the last round can be avoided (and point-to-point channels can be used instead), as shown by Badrinarayanan *et al.* [BMMR21]. Moreover, assuming a PKI, the protocol of [GLS15] can be compressed to only two rounds.

For $n = 3$ and $t = 1$, Patra and Ravi [PR18] present a tight upper bound achieving unanimous abort in the setting with point-to-point channels and a broadcast channel. The protocol leverages garbled circuits, (equivocal) non-interactive commitment scheme and a PRG. In the same honest majority setting but for arbitrary n, Ananth *et al.* [ACGJ18] build four variants of a two-round protocol. Two of these variants are in the plain model (without setup), with both point-to-point channels and broadcast available in both rounds. The first

achieves security with unanimous abort and relies on one-way functions, and the second achieves guaranteed output delivery against fail-stop adversaries and additionally relies on semi-honest oblivious transfer. Their other two protocols require a PKI; and achieve guaranteed output delivery against fail-stop and semi-malicious adversaries.

Finally, Cohen *et al.* [CGZ20] present a complete characterization of the feasibility landscape of two-round MPC in the dishonest majority setting, for all broadcast patterns. In particular, they show protocols (without a PKI) for the cases of point-to-point communication in both rounds, point-to-point in the first round and broadcast in the second round, and broadcast in both rounds. The protocols achieve security with selective abort, unanimous abort and identifiable abort, respectively. All protocols rely on two-round oblivious transfer.

2 Secure Multiparty Computation (MPC) Definitions

2.1 Security Model

We follow the real/ideal world simulation paradigm and we adopt the security model of Cohen, Garay and Zikas [CGZ20]. As in their work, we state our results in a stand-alone setting.[5]

Real-world. An n-party protocol $\Pi = (P_1, \ldots, P_n)$ is an n-tuple of probabilistic polynomial-time (PPT) interactive Turing machines (ITMs), where each party P_i is initialized with input $x_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$. We let $\mathcal{A}$ denote a special PPT ITM that represents the adversary and that is initialized with input that contains the identities of the corrupt parties, their respective private inputs, and an auxiliary input. The protocol is executed in rounds (i.e., the protocol is synchronous), where each round consists of the send phase and the receive phase, where parties can respectively send the messages from this round to other parties and receive messages from other parties. In every round parties can communicate either over a broadcast channel or a fully connected point-to-point (P2P) network, where we additionally assume all communication to be private and ideally authenticated. (Given a PKI and a broadcast channel, such a fully connected point-to-point network can be instantiated.)

During the execution of the protocol, the corrupt parties receive arbitrary instructions from the adversary $\mathcal{A}$, while the honest parties faithfully follow the instructions of the protocol. We consider the adversary $\mathcal{A}$ to be rushing, i.e., during every round the adversary can see the messages the honest parties sent before producing messages from corrupt parties.

At the end of the protocol execution, the honest parties produce output, the corrupt parties produce no output, and the adversary outputs an arbitrary function of its view. The view of a party during the execution consists of its input, random coins and the messages it sees during the execution.

---

[5] We note that our security proofs can translate to an appropriate (synchronous) composable setting with minimal changes.

Definition 1 (Real-world execution). *Let $\Pi = (P_1, \ldots, P_n)$ be an n-party protocol and let $\mathcal{I} \subseteq [n]$, of size at most t, denote the set of indices of the parties corrupted by $\mathcal{A}$. The joint execution of Π under $(\mathcal{A}, \mathcal{I})$ in the real world, on input vector $x = (x_1, \ldots, x_n)$, auxiliary input aux and security parameter λ, denoted $\mathrm{REAL}_{\Pi, \mathcal{I}, \mathcal{A}(\mathsf{aux})}(x, \lambda)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}(\mathsf{aux})$ resulting from the protocol interaction.*

Ideal-world. We describe ideal world executions with selective abort (sl-abort), unanimous abort (un-abort), identifiable abort (id-abort), fairness (fairness) and guaranteed output delivery (god).

Definition 2 (Ideal Computation). *Consider* type $\in \{\mathsf{sl\text{-}abort}, \mathsf{un\text{-}abort}, \mathsf{id\text{-}abort}, \mathsf{fairness}, \mathsf{god}\}$. *Let* $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ *be an n-party function and let* $\mathcal{I} \subseteq [n]$, *of size at most t, be the set of indices of the corrupt parties. Then, the joint ideal execution of f under $(\mathcal{S}, \mathcal{I})$ on input vector $x = (x_1, \ldots, x_n)$, auxiliary input* aux *to $\mathcal{S}$ and security parameter λ, denoted* $\mathrm{IDEAL}_{f, \mathcal{I}, \mathcal{S}(\mathsf{aux})}^{\mathsf{type}}(x, \lambda)$, *is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{S}$ resulting from the following ideal process.*

1. Parties send inputs to trusted party*: An honest party P_i sends its input x_i to the trusted party. The simulator $\mathcal{S}$ may send to the trusted party arbitrary inputs for the corrupt parties. Let x_i' be the value actually sent as the input of party P_i.*

2. Trusted party speaks to simulator*: The trusted party computes $(y_1, \ldots, y_n) = f(x_1', \ldots, x_n')$. If there are no corrupt parties or* type $=$ god*, proceed to step 4.*
   (a) *If* type $\in \{\mathsf{sl\text{-}abort}, \mathsf{un\text{-}abort}, \mathsf{id\text{-}abort}\}$*: The trusted party sends $\{y_i\}_{i \in \mathcal{I}}$ to $\mathcal{S}$.*
   (b) *If* type $=$ fairness*: The trusted party sends* ready *to $\mathcal{S}$.*

3. Simulator $\mathcal{S}$ responds to trusted party*:*
   (a) *If* type $=$ sl-abort*: The simulator $\mathcal{S}$ can select a set of parties that will not get the output as $\mathcal{J} \subseteq [n] \setminus \mathcal{I}$. (Note that $\mathcal{J}$ can be empty, allowing all parties to obtain the output.) It sends $(\mathtt{abort}, \mathcal{J})$ to the trusted party.*
   (b) *If* type $\in \{\mathsf{un\text{-}abort}, \mathsf{fairness}\}$*: The simulator can send* abort *to the trusted party. If it does, we take $\mathcal{J} = [n] \setminus \mathcal{I}$.*
   (c) *If* type $=$ id-abort*: If it chooses to abort, the simulator $\mathcal{S}$ can select a corrupt party $i^* \in \mathcal{I}$ who will be blamed, and send $(\mathtt{abort}, i^*)$ to the trusted party. If it does, we take $\mathcal{J} = [n] \setminus \mathcal{I}$.*

4. Trusted party answers parties*:*
   (a) *If the trusted party got* abort *from the simulator $\mathcal{S}$,*
       i. *It sets the abort message* abortmsg*, as follows:*
          − *if* type $\in \{\mathsf{sl\text{-}abort}, \mathsf{un\text{-}abort}, \mathsf{fairness}\}$*, we let* abortmsg $= \perp$.
          − *if* type $=$ id-abort*, we let* abortmsg $= (\perp, i^*)$.
       ii. *The trusted party then sends* abortmsg *to every party P_j, $j \in \mathcal{J}$, and y_j to every party P_j, $j \in [n] \setminus \mathcal{J}$.*
       Note that, if type $=$ god*, we will never be in this setting, since $\mathcal{S}$ was not allowed to ask for an abort.*

(b) Otherwise, it sends y to every P_j, $j \in [n]$.

5. Outputs*: Honest parties always output the message received from the trusted party while the corrupt parties output nothing. The simulator $\mathcal{S}$ outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in \mathcal{I}}$, the messages received by the corrupt parties from the trusted party and its auxiliary input.*

Security Definitions. We now define the security notion for protocols.

Definition 3. *Consider* type $\in \{$sl-abort, un-abort, id-abort, fairness, god$\}$. *Let f : $(\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party function. A protocol Π t-securely computes the function f with* type *security if for every PPT real-world adversary $\mathcal{A}$ there exists a PPT simulator $\mathcal{S}$ such that for every $\mathcal{I} \subseteq [n]$ of size at most t, it holds that*

$$\left\{ \text{REAL}_{\Pi,\mathcal{I},\mathcal{A}(\mathsf{aux})}(x,\lambda) \right\}_{x \in (\{0,1\}^*)^n, \lambda \in \mathbb{N}} \overset{c}{\equiv} \left\{ \text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathcal{S}(\mathsf{aux})}(x,\lambda) \right\}_{x \in (\{0,1\}^*)^n, \lambda \in \mathbb{N}}.$$

2.2 Notation

In this paper, we focus on two-round secure computation protocols. Rather than viewing a protocol Π as an n-tuple of interactive Turing machines, it is convenient to view each Turing machine as a sequence of three algorithms: $\mathtt{frst\text{-}msg}_i$, to compute P_i's first messages to its peers; $\mathtt{snd\text{-}msg}_i$, to compute P_i's second messages; and $\mathtt{output}_i$, to compute P_i's output. Thus, a protocol Π can be defined as $\{(\mathtt{frst\text{-}msg}_i, \mathtt{snd\text{-}msg}_i, \mathtt{output}_i)\}_{i \in [n]}$.

The syntax of the algorithms is as follows:

- $\mathtt{frst\text{-}msg}_i(x_i, r_i) \to (\mathsf{msg}^1_{i \to 1}, \ldots, \mathsf{msg}^1_{i \to n})$ produces the first-round messages of party P_i to all parties. Note that a party's message to itself can be considered to be its state.
- $\mathtt{snd\text{-}msg}_i(x_i, r_i, \mathsf{msg}^1_{1 \to i}, \ldots, \mathsf{msg}^1_{n \to i}) \to (\mathsf{msg}^2_{i \to 1}, \ldots, \mathsf{msg}^2_{i \to n})$ produces the second-round messages of party P_i to all parties.
- $\mathtt{output}_i(x_i, r_i, \mathsf{msg}^1_{1 \to i}, \ldots, \mathsf{msg}^1_{n \to i}, \mathsf{msg}^2_{1 \to i}, \ldots, \mathsf{msg}^2_{n \to i}) \to y_i$ produces the output returned to party P_i.

When the first round is over broadcast channels, we consider $\mathtt{frst\text{-}msg}_i$ to return only one message — msg^1_i. Similarly, when the second round is over broadcast channels, we consider $\mathtt{snd\text{-}msg}_i$ to return only msg^1_i.

Throughout our negative results, we omit the randomness r, and instead focus on deterministic protocols, modeling the randomness implicitly as part of the algorithm.

3 No Broadcast: Impossibility of Unanimous Abort

For our negative results in the setting where no broadcast is available, we leverage related negative results for broadcast (or byzantine agreement). To show that

guaranteed output delivery is impossible in two rounds of peer-to-peer communication, we can use the fact that broadcast cannot be realized in two rounds for $t > 1$ [FL82,DS83]. To show the impossibility of weaker guarantees such as unanimous abort in this setting, we prove that a weaker flavor of broadcast, called *(weak) detectable* broadcast [FGMv02] — where all parties either learn the broadcast bit, or unanimously abort — cannot be realized in two rounds either.

We state the definitions of *broadcast* and *detectable broadcast* (from Fitzi *et al.* [FGMv02]) below.

Definition 4 (Broadcast). *A protocol among n parties, where the dealer $D = P_1$ holds an input value $x \in \{0, 1\}$ and every other party $P_i, i \in [2, \ldots, n]$ outputs a value $y_i \in \{0, 1\}$, achieves* broadcast *if it satisfies the following two conditions:*

Validity: *If the dealer D is honest then all honest parties P_i output $y_i = x$.*
Consistency: *All honest parties output the same value $y_2 = \cdots = y_n = y$.*

Definition 5 (Detectable Broadcast). *A protocol among n parties achieves* detectable broadcast *if it satisfies the following three conditions:*

Correctness: *All honest parties unanimously accept or unanimously reject the protocol. If all honest parties accept then the protocol achieves broadcast.*
Completeness: *If all parties are honest then all parties accept.*
Fairness: *If any honest party rejects the protocol then the adversary gets no information about the dealer's input x.*

We additionally define *weak detectable broadcast*.

Definition 6 (Weak Detectable Broadcast). *A protocol among n parties achieves* weak detectable broadcast *if it satisfies only the correctness and completeness requirements of detectable broadcast.*

An alternative way of viewing broadcast, through the lens of secure computation, is by considering the simple broadcast function $f_{\mathsf{bc}}(x, \bot, \ldots, \bot) = (\bot, x, \ldots, x)$ which takes an input bit x from the dealer $D = P_1$, and outputs that bit to all other parties. *Broadcast* (Definition 4) is exactly equivalent to computing f_{bc} with guaranteed output delivery; *detectable broadcast* (Definition 5) is equivalent to computing it with fairness; and *weak detectable broadcast* (Definition 6) is equivalent to computing it with unanimous abort.

Theorem 1. *Weak detectable broadcast cannot be achieved in two rounds of peer-to-peer communication for corruption threshold $t > 1$.*

Proof. We prove Thm 1 by contradiction. We let

$$\Pi_{\mathtt{wdbc}} = \{(\mathtt{frst\text{-}msg}_i, \mathtt{snd\text{-}msg}_i, \mathtt{output}_i)\}_{i \in [1, \ldots, n]}$$

be the description of the two-round weak detectable broadcast protocol. We use the notation we introduce for two-round secure computation in Section 2.2, and

12

consider the weak detectable broadcast protocol to be a secure computation with unanimous abort of f_{bc}. We let $x_1 = x$ denote the bit being broadcast by the dealer $D = P_1$, and $x_i = \bot$ for $i \in [2, \ldots, n]$ be placeholders for other parties' inputs. We assume that $\mu = negl$ is the negligible probability with which security of $\Pi_{\mathtt{wdbc}}$ fails.

Below we consider an execution of $\Pi_{\mathtt{wdbc}}$ and a sequence of scenarios involving different adversarial strategies with two corruptions ($t = 2$). The dealer $D = P_1$ is corrupt in all of these; at most one of the receiving parties $P_2, \ldots, P_n$ is corrupt at a time. We argue that each subsequent strategy clearly requires certain parties to output certain values, by the definition of weak detectable broadcast. In the last strategy, we see a contradiction, where some parties must output both 0 and 1. Therefore, $\Pi_{\mathtt{wdbc}}$ could not have been a weak detectable broadcast protocol. In all of the strategies below, we let $\mathsf{msg}_{b,i \to j}$ denote a party P_i's bth-round message to party P_j; we only specify how these messages are generated when this is done dishonestly.

Scenario 1: D is corrupt.
    Round 1: D behaves honestly using input $x = 0$.
    Round 2: D behaves honestly using input $x = 0$.
By completeness (which holds since everyone behaved honestly), all honest parties must accept the protocol. By correctness, the protocol must thus achieve broadcast. By validity, all honest parties must output 0. Since completeness, correctness and validity hold with probability at least $1 - \mu$, we can infer that honest parties must output 0 with probability at least $1 - \mu$.

Scenario 2_M: D and P_2 are corrupt.
    Round 1: D computes two different sets of messages, using different inputs $x = 0$ and $x = 1$, as follows:

$$(\mathsf{msg}_{1 \to 1}^{1,(0)}, \ldots, \mathsf{msg}_{1 \to n}^{1,(0)}) \leftarrow \mathtt{frst\text{-}msg}_1(x = 0)$$

$$(\mathsf{msg}_{1 \to 1}^{1,(1)}, \ldots, \mathsf{msg}_{1 \to n}^{1,(1)}) \leftarrow \mathtt{frst\text{-}msg}_1(x = 1)$$

D sends $\mathsf{msg}_{1 \to 3}^{1,(0)}, \ldots, \mathsf{msg}_{1 \to n}^{1,(0)}$ to parties $P_3, \ldots, P_n$. P_2 behaves honestly.
    Round 2: D behaves honestly using input $x = 0$. P_2 computes two different sets of second-round messages, as follows:

$$(\mathsf{msg}_{2 \to 1}^{2,(0)}, \ldots, \mathsf{msg}_{2 \to n}^{2,(0)}) \leftarrow \mathtt{snd\text{-}msg}_2(\bot, \mathsf{msg}_{1 \to 2}^{1,(0)}, \mathsf{msg}_{2 \to 2}^1, \ldots, \mathsf{msg}_{n \to 2}^1)$$

$$(\mathsf{msg}_{2 \to 1}^{2,(1)}, \ldots, \mathsf{msg}_{2 \to n}^{2,(1)}) \leftarrow \mathtt{snd\text{-}msg}_2(\bot, \mathsf{msg}_{1 \to 2}^{1,(1)}, \mathsf{msg}_{2 \to 2}^1, \ldots, \mathsf{msg}_{n \to 2}^1)$$

P_2 sends $\mathsf{msg}_{2 \to n}^{2,(1)}$ to P_n (pretending, essentially, that D dealt a 1), and $\mathsf{msg}_{2 \to i}^{2,(0)}$ to other parties P_i (pretending that D dealt a 0).
$P_3, \ldots, P_{n-1}$ must accept and output 0 with probability at least $1 - \mu$, since their views are identical to those in the previous scenario. By correctness, P_n must also accept when other honest parties accept. By consistency, P_n must also output 0. Since correctness or consistency break with probability at most μ, P_n outputs 0 with probability at least $1 - 2\mu$.

Scenario 2_H**:** D is corrupt.

    Round 1: D sends $\mathsf{msg}_{1\to2}^{1,(1)}$ to P_2, and $\mathsf{msg}_{1\to i}^{1,(0)}$ to other parties P_i.

    Round 2: D continues to represent $x=1$ towards P_2 and $x=0$ towards the others.

P_n must accept and output 0 with probability at least $1-2\mu$, since its view is the same as in the previous scenario. By correctness, $P_2,\ldots,P_{n-1}$ must also accept when P_n accepts. By consistency, $P_2,\ldots,P_{n-1}$ must also output 0. Since correctness or consistency break with probability at most μ, $P_2,\ldots,P_{n-1}$ output 0 with probability at least $1-3\mu$.

Now, skipping ahead, we generalize, for $k\in[3,\ldots,n-1]$:

Scenario k_M**:** D and P_k are corrupt.

    Round 1: D sends $\mathsf{msg}_{1\to i}^{1,(1)}$ to $P_2,\ldots,P_{k-1}$, and $\mathsf{msg}_{1\to i}^{1,(0)}$ to the other parties $P_{k+1},\ldots,P_n$. P_k acts honestly.

    Round 2: D continues to represent $x=1$ to $P_2,\ldots,P_{k-1}$ and $x=0$ to $P_{k+1},\ldots,P_n$. In the second round P_k acts analogously to P_2 in scenario 2_M; i.e., P_k uses $\mathsf{msg}_{1\to k}^{1,(0)}$ to compute $(\mathsf{msg}_{k\to1}^{2,(0)},\ldots,\mathsf{msg}_{k\to n-1}^{2,(0)})$ (which it sends to $P_2,\ldots,P_{n-1}$), and $\mathsf{msg}_{1\to k}^{1,(1)}$ to compute $\mathsf{msg}_{k\to n}^{2,(1)}$ (which it sends to P_n).

$P_2,\ldots,P_{n-1}$ must accept and output 0 with probability at least $1-(2(k-1)-1)\mu = 1-(2k-3)\mu$, since their views are identical to those in the previous scenario (namely Scenario $(k-1)_H$). By correctness, P_n must also accept when other honest parties accept. By consistency, P_n must also output 0. Since correctness or consistency break with probability at most μ, P_n outputs 0 with probability at least $1-(2k-3)\mu-\mu=1-2(k-1)\mu$.

Scenario k_H**:** D is corrupt.

    Round 1: D sends $\mathsf{msg}_{1\to i}^{1,(1)}$ to $P_2,\ldots,P_k$, and $\mathsf{msg}_{1\to i}^{1,(0)}$ to the other parties $P_{k+1},\ldots,P_n$.

    Round 2: D continues to represent $x=1$ to $P_2,\ldots,P_k$ and $x=0$ to $P_{k+1},\ldots,P_n$.

P_n must accept and output 0 with probability at least $1-2(k-1)\mu$, since its view is the same as in the previous scenario. By correctness, $P_2,\ldots,P_{n-1}$ must also accept. By consistency, $P_2,\ldots,P_{n-1}$ must also output 0. Since correctness or consistency break with probability at most μ, $P_2,\ldots,P_{n-1}$ output 0 with probability at least $1-2(k-1)\mu-\mu=1-(2k-1)\mu$.

We end with Scenarios n_M,n_H.

Scenario n_M**:** D and P_n are corrupt.

    Round 1: D behaves honestly using input $x=1$. P_n behaves honestly.

    Round 2: D behaves honestly using input $x=1$. P_n pretends D dealt a 0 towards, e.g., only P_2. More precisely, P_n uses $\mathsf{msg}_{1\to n}^{1,(0)}$ to compute $\mathsf{msg}_{n\to2}^{2,(0)}$ (which it sends to P_2), and $\mathsf{msg}_{1\to n}^{1,(1)}$ to compute $(\mathsf{msg}_{n\to3}^{2,(1)},\ldots,\mathsf{msg}_{n\to n-1}^{2,(1)})$ (which it sends to $P_3,\ldots,P_{n-1}$).

P_2 must accept and output 0 with probability at least $1 - (2(n-1) - 1)\mu = 1 - (2n - 3)\mu$, since its view is the same as in the previous scenario (namely, Scenario $(n-1)_H$). By correctness, $P_3, \ldots, P_{n-1}$ must also accept. By consistency, $P_3, \ldots, P_{n-1}$ must also output 0. This must happen with probability at least $1 - (2n - 3)\mu - \mu = 1 - 2(n - 1)\mu$.

Scenario n_H: D is corrupt.

   Round 1: D behaves honestly using input $x = 1$.
   Round 2: D behaves honestly using input $x = 1$.

In Scenario n_H, on the one hand, by completeness (which holds as everyone behaved honestly), all honest parties must accept the protocol; by validity, all honest parties must output 1. On the other hand, since the view of $P_3, \ldots, P_{n-1}$ is the same as their respective views in the previous scenario, they must output 0 with probability at least $1 - 2(n - 1)\mu$, which is overwhelming. This is a contradiction.

The impossibility of realizing weak detectable broadcast in two rounds for $t > 1$ clearly implies that there exists a function (specifically, $f_{\tt bc}$) which is impossible to compute with unanimous abort for $t > 1$ in two rounds of peer-to-peer communication.

Corollary 1 (P2P-P2P, UA, $t > 1$). *There exist functions f such that no n-party two-round protocol can compute f with unanimous abort against $t > 1$ corruptions in two rounds of peer-to-peer communication.*

4 Broadcast in the Second Round: Impossibility of Fairness

In this section, we show that it is not possible to design fair protocols tolerating $t > 1$ corruptions when broadcast is available only in the second round.

Theorem 2 (P2P-BC, FAIR, $t > 1$). *There exist functions f such that no n-party two-round protocol can compute f with fairness against $t > 1$ corruptions while making use of broadcast only in the second round (i.e. where the first round is over point-to-point channels and second round uses both broadcast and point-to-point channels).*

In our proof we use the function $f_{\tt mot}$, which is defined below. Let P_1 hold as input a bit $X_1 = b \in \{0, 1\}$, and every other party P_i ($i \in \{2, \ldots, n\}$) hold as input a pair of strings, denoted as $X_i = (x_i^0, x_i^1)$.

$$f_{\tt mot}\big(X_1 = b, X_2 = (x_2^0, x_2^1), \ldots, X_n = (x_n^0, x_n^1)\big) = (x_2^b, x_3^b, \ldots, x_n^b).$$

Proof. We prove Thm 2 by contradiction. Let Π be a protocol that computes $f_{\tt mot}$ with fairness by using broadcast only in the second round. Consider an execution of Π where X_i denotes the input of P_i. We describe a sequence of scenarios $C_1, \ldots, C_n, C_n^*$. In each scenario, P_1 and at most one other party is

corrupt. In all the scenarios, the corrupt parties behave honestly (in particular, they use their honest inputs), but may drop incoming or outgoing messages.

At a high-level, the sequence of scenarios is designed so that corrupt P_1 drops her first-round message to one additional honest party in each scenario. We show that in each scenario, the adversary manages to obtain the output computed with respect to $X_1 = b$ and (at least some of) the honest parties' inputs. This leads to a contradiction, because the final scenario involves no first-round messages from P_1 related to its input $X_1 = b$, but the adversary is still able to learn x_i^b corresponding to some honest P_i. In particular, this implies that the adversary is able to re-compute second-round messages from P_1 with different choices of input X_1, obtaining multiple outputs (on different inputs).

Before describing the scenarios in detail, we define some useful notation. Let $(X_1, \ldots, X_n)$ denote a specific combination of inputs that are fixed across all scenarios. Let $\mu = negl$ denote the negligible probability with which the security of Π breaks. We assume, without loss of generality, that the second round of Π involves broadcast communication alone (as given a PKI and a broadcast channel, point-to-point communication can be realized by broadcasting encryptions of the private messages using the public key of the recipient). Let $\widetilde{\mathsf{msg}}_i^2$ denote P_i's second-round broadcast message, computed honestly given that P_i did not receive the private message (i.e. the communication over point-to-point channel) from P_1 in the first round.

Scenario C_1: P_1 is corrupt.
> **Round 1:** P_1 behaves honestly (i.e. follows the instructions of Π).
> **Round 2:** P_1 behaves honestly.

Since everyone behaved honestly, it follows from correctness that P_1 obtains the output $y = f_{\mathsf{mot}}(x_1, \ldots, x_n) = (x_2^b, x_3^b, \ldots, x_n^b)$ with probability at least $1 - \mu$.

Scenario C_2: P_1 and P_2 are corrupt.
> **Round 1:** P_1 and P_2 behave honestly.
> **Round 2:** P_1 remains silent. P_2 pretends she did not receive a first-round message from P_1. In more detail, P_2 sends $\widetilde{\mathsf{msg}}_2^2$ over broadcast channel.

The adversary's view subsumes her view in the previous scenario, so the adversary learns the output $y = (x_2^b, x_3^b, \ldots, x_n^b)$ which allows her to learn x_i^b corresponding to each honest P_i. It follows from the security of Π that honest parties also obtain x_i^b corresponding to each honest P_i (i.e. for $i \in [n] \setminus \{1, 2\}$) with probability at least $1 - \mu$. If not, then either correctness or fairness is violated, which contradicts our assumption that Π is secure.

Scenario C_3: P_1 and P_3 are corrupt.
> **Round 1:** P_1 behaves honestly, but does not send a message to P_2. P_3 behaves honestly.
> **Round 2:** P_1 remains silent. P_3 pretends that she did not receive a first-round message from P_1 (i.e. she sends $\widetilde{\mathsf{msg}}_3^2$ via broadcast).

The adversary's view subsumes the view of an honest P_3 in Scenario C_2 (which includes $\widetilde{\mathsf{msg}}_2^2$); so, the adversary learns $\{x_i^b\}_{i \in [n] \setminus \{1,2\}}$ with probability at least $1 - \mu$. By the fairness of Π, when the adversary obtains this information, honest parties $P_2, P_4, P_5, \ldots, P_n$ must also learn x_i^b corresponding to each honest P_i (i.e. for $i \in [n] \setminus \{1,3\}$).[6] Since the fairness of Π breaks with probability at most μ, parties $P_2, P_4, P_5, \ldots, P_n$ learn $\{x_i^b\}_{i \in [n] \setminus \{1,3\}}$ with probability at least $1 - 2\mu$.

> **Scenario C_4: P_1 and P_4 are corrupt.**
> **Round 1:** P_1 behaves honestly, except that she does not send a message to P_2 and P_3. P_4 behaves honestly.
> **Round 2:** P_1 remains silent. P_4 pretends that she did not receive a first-round message from P_1 (i.e. she sends $\widetilde{\mathsf{msg}}_4^2$ via broadcast).

The adversary's view subsumes the view of an honest P_4 in Scenario C_3 (which includes $\widetilde{\mathsf{msg}}_j^2$, where $j \in \{2,3\}$). Therefore, the adversary learns $\{x_i^b\}_{i \in [n] \setminus \{1,3\}}$ with probability at least $1 - 2\mu$. By the security of Π, honest $P_2, P_3, P_5, \ldots, P_n$ must also obtain x_i^b corresponding to each honest P_i (i.e. for $i \in [n] \setminus \{1,4\}$). Since the security of Π breaks with probability at most μ, parties $P_2, P_3, P_5, \ldots, P_n$ learn $\{x_i^b\}_{i \in [n] \setminus \{1,4\}}$ with probability at least $1 - 3\mu$.

Generalizing the above for $k = 3$ to n:

> **Scenario C_k: P_1 and P_k are corrupt.**
> **Round 1:** P_1 behaves honestly, except that she does not send a message to $P_2, P_3, \ldots, P_{k-1}$. P_k behaves honestly.
> **Round 2:** P_1 remains silent. P_k pretends that she did not receive a first-round message from P_1 (i.e. she sends $\widetilde{\mathsf{msg}}_k^2$ via broadcast).

The adversary's view subsumes the view of an honest P_k in Scenario C_{k-1} (which includes messages $\widetilde{\mathsf{msg}}_j^2$, where $j \in \{2, \ldots, k-1\}$). Thus, the adversary learns $\{x_i^b\}_{i \in [n] \setminus \{1,k-1\}}$ with probability at least $1 - (k-2)\mu$. By the security of Π, honest parties should obtain x_i^b corresponding to each honest P_i (i.e. for $i \in [n] \setminus \{1,k\}$). Since the security of Π breaks with probability at most μ, honest parties learn the values x_i^b with probability at least $1 - (k-2)\mu - \mu = 1 - (k-1)\mu$.

Finally, we describe the last scenario:

> **Scenario C_n^*: P_1 and P_n are corrupt.**
> **Round 1:** P_1 remains silent. P_n behaves honestly.
> **Round 2:** P_1 and P_n remain silent.

The adversary's view subsumes her view in Scenario C_n (which includes messages $\widetilde{\mathsf{msg}}_j^2$, where $j \in \{1, \ldots, n-1\}$). Thus, in Scenario C_n^*, the adversary is able to learn $\{x_i^b\}_{i \in [n] \setminus \{1,n-1\}}$ with probability at least $1 - (n-1)\mu$. This leads us to the

---

[6] Note that we conclude that the honest parties learn x_2^b, which the adversary may, for some reason, not have learned. This is because in the ideal functionality, output is considered as a single unit of information; fairness requires that if the adversary learns *any* output it could not have obtained solely from its own inputs, then the honest parties must learn the *entire* output.

final contradiction: C_n^* does not involve any message from P_1 related to the input $X_1 = b$, but the adversary was able to obtain $\{x_i^b\}_{i \in [n]\setminus\{1,n-1\}}$. This implies that the adversary can compute $\{x_i^{b'}\}_{i \in [n]\setminus\{1,n-1\}}$ with respect to any input $X_1 = b'$ of her choice. This "residual attack" breaks the privacy property of the protocol, as it allows the adversary to learn both input strings of an honest P_i. (which is not allowed as per the ideal realization of $f_{\texttt{mot}}$).

Lastly, we note that the above proof requires that the function computed is such that each party receives the output. This is because the inference in Scenario C_k ($k \in [n]$) relies on the adversary obtaining output on behalf of P_k.

5 Completing the Picture: Impossibility Results for $n \leq 3t$

In the previous two sections, we showed the impossibility of unanimous abort when no broadcast is available, and the impossibility of fairness when broadcast is only available in the second round. However, both of those impossibility results only hold for $t > 1$. In this section, using different techniques, we extend those results to the case when $t = 1$ and $n = 3$. In our impossibility results in this section, we use a property which we call *last message resiliency*.

Definition 7 (Last Message Resiliency). *A protocol is t-last message resilient if, in an honest execution, any protocol participant P_i can compute its output without using t of the messages it received in the last round.*

More formally, consider a protocol $\Pi = \{(\texttt{frst-msg}_i, \texttt{snd-msg}_i, \texttt{output}_i)\}_{i \in [1,\ldots,n]}$. The protocol is t-last message resilient if, for each $i \in [1,\ldots,n]$ and each $S \subseteq \{1,\ldots,n\}\setminus\{i\}$ such that $|S| \leq t$, the output function $\texttt{output}_i$ returns the correct output even without second round messages from parties $P_i, i \in S$. That is, for all security parameters λ, for all sets $S \subseteq \{1,\ldots,n\}\setminus\{i\}$ such that $|S| \leq t$, for all inputs $x_1, \ldots, x_n$,

$$\Pr\left(\begin{matrix} \texttt{output}_i(x_i, \texttt{msg}^1_{1 \to i}, \ldots, \texttt{msg}^1_{n \to i}, \tilde{\texttt{msg}}^2_{1 \to i}, \ldots, \tilde{\texttt{msg}}^2_{n \to i}) \\ \neq \texttt{output}_i(x_i, \texttt{msg}^1_{1 \to i}, \ldots, \texttt{msg}^1_{n \to i}, \texttt{msg}^2_{1 \to i}, \ldots, \texttt{msg}^2_{n \to i}) \end{matrix}\right) = negl(\lambda)$$

over the randomness used in the protocol, where, for $j \in [1,\ldots,n]$,

$$(\texttt{msg}^1_{j \to 1}, \ldots, \texttt{msg}^1_{j \to n}) \leftarrow \texttt{frst-msg}_j(x_j),$$

$$(\texttt{msg}^2_{j \to 1}, \ldots, \texttt{msg}^2_{j \to n}) \leftarrow \texttt{snd-msg}_j(x_j, \texttt{msg}^1_{1 \to j}, \ldots, \texttt{msg}^1_{n \to j}),$$

and

$$\tilde{\texttt{msg}}^2_{j \to i} = \begin{cases} \texttt{msg}^2_{j \to i}, & \text{if } j \notin S, \\ \perp & \text{otherwise.} \end{cases}$$

Theorem 3. *Any protocol Π which achieves secure computation with unanimous abort with corruption threshold t and whose last round can be executed over peer-to-peer channels must be t-last message resilient.*

Proof. We prove this by contradiction. Assume Π achieves unanimous abort, and is not t-resilient. Then, by definition, there exist inputs $x_1, \ldots, x_n$, an $i \in [1, \ldots, n]$ and a subset $S \subseteq \{1, \ldots, n\} \backslash \{i\}$ (such that $|S| \leq t$) where, with non-negligible probability,

$$\texttt{output}_i(x_i, \texttt{msg}^1_{1 \to i}, \ldots, \texttt{msg}^1_{n \to i}, \tilde{\texttt{msg}}^2_{1 \to i}, \ldots, \tilde{\texttt{msg}}^2_{n \to i})$$
$$\neq \texttt{output}_i(x_i, \texttt{msg}^1_{1 \to i}, \ldots, \texttt{msg}^1_{n \to i}, \texttt{msg}^2_{1 \to i}, \ldots, \texttt{msg}^2_{n \to i})$$

(where the messages are produced in the way described in Definition 7).

The adversary can use this by corrupting P_j, $j \in S$; it will behave honestly, except in the last round, where $P_j, j \in S$ will not send messages to P_i. (Note that the ability to send last round messages to some parties but not others relies on the fact that the last round is over peer-to-peer channels.) With non-negligible probability, P_i will receive an incorrect output (e.g. an abort). However, this cannot occur in a protocol with unanimous abort; all other honest parties must accept the protocol and produce the correct output (since their views are the same as in an entirely honest execution), so P_i must as well.

Theorem 4. *Any protocol Π which achieves secure computation with fairness with corruption threshold t must be t-last message resilient.*

Proof. We prove this by contradiction. Assume Π achieves fairness, and is not t-resilient. Then, by definition, there exist inputs $x_1, \ldots, x_n$, an $i \in [1, \ldots, n]$ and a subset $S \subseteq \{1, \ldots, n\} \backslash \{i\}$ (such that $|S| \leq t$) where, with non-negligible probability,

$$\texttt{output}_i(x_i, \texttt{msg}^1_{1 \to i}, \ldots, \texttt{msg}^1_{n \to i}, \tilde{\texttt{msg}}^2_{1 \to i}, \ldots, \tilde{\texttt{msg}}^2_{n \to i})$$
$$\neq \texttt{output}_i(x_i, \texttt{msg}^1_{1 \to i}, \ldots, \texttt{msg}^1_{n \to i}, \texttt{msg}^2_{1 \to i}, \ldots, \texttt{msg}^2_{n \to i}).$$

(where the messages are produced in the way described in Definition 7).

The adversary can use this by corrupting P_j, $j \in S$. As in the previous proof, it will behave honestly, except in the last round, where $P_j, j \in S$ will not send messages to P_i. With non-negligible probability, P_i will receive an incorrect output (e.g. an abort), while the rushing adversary will learn the output, since it will have all of the messages it would have gotten in a fully honest execution of the protocol. This violates fairness.[7]

Theorem 5. *There exists a function f such that any protocol Π securely realizing f with corruption threshold t such that $n \leq 3t$ and whose first round can be executed over peer-to-peer channels cannot be t-last message resilient.*

Proof. Consider the function $f_{\texttt{mot}}$ described in the proof of Thm 2, where party P_1 provides as input a choice bit $X_1 = b \in \{0, 1\}$ and every other party P_i provides as input a pair of strings i.e. $X_i = (x_i^0, x_i^1)$.

---

[7] Note that while P_i does not learn the output, other honest parties might. However, even one honest party not receiving the output is a violation of fairness if the adversary learns the output.

Consider an adversary corrupting P_1. The adversary should clearly be unable to recompute the function with multiple inputs, e.g., with respect to both $X_1 = 0$ and $X_1 = 1$ (as this will allow it to learn both the input strings of the honest parties which is in contrast to an ideal execution, where it can learn exactly one of the input strings).

We now show that, in a t-last message resilient (where $n \leq 3t$) two-round protocol Π where the first round is over peer-to-peer channels, P_1 can always learn both of those outputs. Consider a corrupt P_1, and partition the honest parties into two sets of equal size (assuming for simplicity that the number of honest parties is even): S_0 and S_1. Note that $|S_0| = |S_1| = \frac{n-t}{2} \leq t$.

P_1 uses $X_1 = 0$ to compute its first round messages to S_0; it uses $X_1 = 1$ to compute its first round messages to S_1. (Note that the ability to send first round messages based on different inputs relies on the fact that the first round is over peer-to-peer channels.) All other parties behave honestly. Because the protocol Π is t-last message resilient, and because S_1 contains t or fewer parties, P_1 has enough second round messages excluding those it received from S_1 to compute its output. Note that all second round messages except for those received from S_1 are distributed exactly as in an honest execution with $X_1 = 0$; therefore, by last message resiliency, P_1 learns $(x_2^0, x_3^0, \ldots, x_n^0)$ (as per the definition of $f_{\mathtt{mot}}$). Similarly, by excluding second round messages it received from S_0, P_1 learns the output $(x_2^1, x_3^1, \ldots, x_n^1)$ i.e. the output computed based on $X_1 = 1$. This is clearly a violation of privacy.

Corollary 2 (P2P-P2P, UA, $n \leq 3t$). *Secure computation of general functions with unanimous abort cannot be achieved in two rounds of peer-to-peer communication for corruption threshold t such that $n \leq 3t$.*

This corollary follows directly from Theorems 3 and 5.

Remark 1. Note that for $t > 1$, Cor 2 is subsumed by Cor 1. However, Cor 2 covers the case of $t = 1$ and $n = 3$, closing the question of unanimous abort with honest majority in two rounds of peer-to-peer communication.

Corollary 3 (P2P-BC, FAIR, $n \leq 3t$). *Secure computation of general functions with fairness cannot be achieved in two rounds the first of which is over peer-to-peer channels for corruption threshold t such that $n \leq 3t$.*

This corollary follows from Theorems 4 and 5.

6 Broadcast in the First Round: Guaranteed Output Delivery

In this section, we argue that *any* protocol that achieves guaranteed output delivery in two rounds of broadcast also achieves guaranteed output delivery when broadcast is available in the first round only. We first show that if the protocol achieves guaranteed output delivery with corruption threshold t in two

rounds of broadcast, it achieves the same guarantee with threshold $t-1$ when the second round is over peer-to-peer channels. We next show that if the first-round messages commit corrupt parties to their inputs, the second round can be run over peer-to-peer channels with no loss in corruption budget.

Theorem 6. *Let $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ be a two broadcast-round protocol that securely computes the function f with guaranteed output delivery against an adversary corrupting t parties. Then $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ achieves the same guarantee when the second round is run over peer-to-peer channels but with $t-1$ corruptions.*

Proof (Sketch). Let $\tilde{\Pi}_{\mathsf{bc}}^{\mathsf{god}}$ denote the protocol where the second round is run over peer-to-peer channels but with $t-1$ corruptions. Towards a contradiction, assume $\tilde{\Pi}_{\mathsf{bc}}^{\mathsf{god}}$ is not secure against $(t-1)$ corruptions; in particular, assume that there is an adversary $\tilde{\mathcal{A}}$ that breaks security.

We first observe that $\tilde{\mathcal{A}}$ certainly can't cause honest parties to abort in $\tilde{\Pi}_{\mathsf{bc}}^{\mathsf{god}}$ by sending them incorrect things in the second round, since $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ achieves guaranteed output delivery, meaning that honest parties do not abort no matter what $\tilde{\mathcal{A}}$ does. Therefore, all $\tilde{\mathcal{A}}$ can hope for is to cause disagreement in $\tilde{\Pi}_{\mathsf{bc}}^{\mathsf{god}}$. In particular, $\tilde{\mathcal{A}}$ can send different second-round messages to different honest parties, hoping that honest parties end up with outputs computed on different corrupt party inputs. However, if $\tilde{\mathcal{A}}$ could do that, we could use $\tilde{\mathcal{A}}$ to build an adversary $\mathcal{A}$ that breaks the security of $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ by corrupting one additional honest party, mentally sending different messages to it, and obtaining the output on two different sets of its own inputs.

Suppose $\tilde{\mathcal{A}}$ can make a pair of honest parties in $\tilde{\Pi}_{\mathsf{bc}}^{\mathsf{god}}$ — P_i and P_j — obtain different outputs by sending different second-round messages to them. Then, we construct our adversary $\mathcal{A}$ for $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ as follows. $\mathcal{A}$ corrupts the same $t-1$ parties as $\tilde{\mathcal{A}}$, as well as one additional honest party — P_i — who will behave semi-honestly. $\mathcal{A}$ uses the second-round messages sent by $\tilde{\mathcal{A}}$ to P_j as her broadcast second-round messages in $\Pi_{\mathsf{bc}}^{\mathsf{god}}$. However, $\mathcal{A}$ also computes what P_i's output would have been if she had broadcast the second-round messages sent by $\tilde{\mathcal{A}}$ to P_i. This allows $\mathcal{A}$ to obtain the output on behalf of P_i on two different sets of inputs, breaking the security of $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ (and completing the proof). ∎

Theorem 7. *Let $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ be a two broadcast-round protocol that securely computes the function f with guaranteed output delivery with the additional constraint that a simulator can extract inputs from the first-round messages and it is efficient to check whether a given second-round message is correct. Then $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ achieves the same guarantee when the second round is run over point-to-point channels.*

Proof (Sketch). Starting from the protocol $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ it is possible to define another protocol $\Pi_{\mathsf{bcp2p}}^{\mathsf{god}}$ that has the following modifications: (1) the second round messages of $\Pi_{\mathsf{bc}}^{\mathsf{god}}$ are sent over point-to-point channels and (2) the honest parties compute their output based on all the first round messages and the subset C of second round messages that are generated correctly. (Observe that $|C| \geq n-t$, because at least $n-t$ parties are honest.)

Relying on the GOD security of $\Pi_{\text{bc}}^{\text{god}}$, it is possible to claim that $\Pi_{\text{bcp2p}}^{\text{god}}$ also achieves GOD. This follows from two important observations. First, since the input is extracted from the first round of $\Pi_{\text{bcp2p}}^{\text{god}}$ which is over broadcast, the adversary cannot cause disagreement among the honest parties with respect to her input (i.e. she cannot send messages based on different inputs to different honest parties). Second, in $\Pi_{\text{bcp2p}}^{\text{god}}$ the honest parties are always able to compute the output; otherwise, the honest parties in $\Pi_{\text{bc}}^{\text{god}}$ would not have been able to compute an output when $\mathcal{A}$ does not send any second round message, which contradicts GOD security.

Next, we observe that the two broadcast-round protocol of Gordon *et al.* [GLS15] has the two properties required by Thm 7. The protocol of Gordon *et al.* [GLS15] uses zero knowledge proofs to compile a semi-malicious protocol into a fully malicious one. The zero knowledge proofs accompanying the first round messages can be used for input extraction; the zero knowledge proofs accompanying the second round messages can be used to efficiently determine which of these second round messages are generated correctly.

7 One-or-Nothing Secret Sharing

In Section 8, we will show a protocol that achieves security with identifiable abort in the honest majority setting in two rounds, only the second of which is over broadcast. In this section, we introduce an important building block for that protocol which we call *one-or-nothing secret sharing*.

We define one-or-nothing secret sharing as a new flavor of secret sharing wherein the dealer can share a *vector* of secrets. While traditional secret sharing schemes are designed for receivers to eventually publish their shares and recover the entirety of what was shared, one-or-nothing secret sharing is designed for receivers to eventually recover *at most one* of the shared values. While reconstruction usually requires each party to contribute its entire share, in one-or-nothing secret sharing, each party instead *votes* on the index of the value to reconstruct by producing a "ballot" based on its secret share. If two parties vote for different indices, the set of published ballots should reveal nothing about any of the values. However, some parties are allowed to equivocate — they might be unsure which index they wish to vote for, so they will support the preference of the majority. If a majority votes for the same index, and the rest equivocate, the ballots enable the recovery of the value at that index.

Our secure computation construction in Section 8 uses one-or-nothing secret sharing to share labels for garbled circuits. However, we imagine one-or-nothing secret sharing might be of independent interest, e.g. in voting scenarios where unanimity among the decided voters is important.

7.1 Definitions

Syntax The natural syntax for a one-or-nothing secret sharing scheme consists of a tuple of three algorithms (share, vote, reconstruct).

$\mathtt{share}(x^{(1)}, \dots, x^{(l)}) \to (s, s_1, \dots, s_n)$ is an algorithm that takes l values $x^{(1)},$ $\dots, x^{(l)}$, and produces the secret shares $s_1, \dots, s_n$, as well as the public share s.

$\mathtt{vote}(s, s_i, v) \to \overline{s}_i$ is an algorithm that takes the public share s, a secret share s_i, and a vote v, where $v \in \{1, \dots, l, \perp\}$ can either be an index of a value, or it can be $\perp$ if party i is unsure which value it wants to vote for. It outputs a public ballot $\overline{s}_i$.

$\mathtt{reconstruct}(s, \overline{s}_1, \dots, \overline{s}_n) \to \{x^{(v)}, \perp\}$ is an algorithm that takes the public share s, all of the ballots $\overline{s}_1, \dots, \overline{s}_n$, and outputs either the value $x^{(v)}$ which received a majority of votes, or outputs $\perp$.

Non-Interactive One-or-Nothing Secret Sharing We modify this natural syntax to ensure that each party can vote even if they have not received a secret share. This is important in case e.g. the dealer is corrupt, and chooses not to distribute shares properly. We call such a scheme a *non-interactive* one-or-nothing secret sharing scheme. A non-interactive one-or-nothing secret sharing scheme consists of a tuple of four algorithms $(\mathtt{setup}, \mathtt{share}, \mathtt{vote}, \mathtt{reconstruct})$.

$\mathtt{setup}(1^\lambda) \to \mathtt{sk}$ is an algorithm that produces a key shared between the dealer and one of the receivers. (This can be non-interactively derived by both dealer and receiver by running $\mathtt{setup}$ on randomness obtained from e.g. key exchange.)

$\mathtt{share}(\mathtt{sk}_1, \dots, \mathtt{sk}_n, x^{(1)}, \dots, x^{(l)}) \to s$ is an algorithm that takes the n shared keys $\mathtt{sk}_1, \dots, \mathtt{sk}_n$ and the l values $x^{(1)}, \dots, x^{(l)}$, and produces a public share s.

$\mathtt{vote}(\mathtt{sk}_i, v) \to \overline{s}_i$ is an algorithm that takes a secret share s_i and a vote v, where $v \in \{1, \dots, l, \perp\}$ can either be an index of a value, or it can be $\perp$ if party i is unsure which value it wants to vote for. It outputs a public ballot $\overline{s}_i$.

$\mathtt{reconstruct}(s, \overline{s}_1, \dots, \overline{s}_n) \to \{x^{(v)}, \perp\}$ is an algorithm that takes the public share s, all of the ballots $\overline{s}_1, \dots, \overline{s}_n$, and outputs either the value $x^{(v)}$ which received a majority of votes, or outputs $\perp$.

Security We require three properties of one-or-nothing secret sharing: *correctness*, *privacy* (which requires that if fewer than $t + 1$ parties vote for an index, the value at that index stays hidden) and *contradiction-privacy* (which requires that if two parties vote for different indices, all values stay hidden). Below we define these formally for *non-interactive* one-or-nothing secret sharing.

Definition 8 (One-or-Nothing Secret Sharing: Correctness). *Informally, this property requires that when at least $n - t$ parties produce their ballot using the same v (and the rest produce their ballot with $\perp$), $\mathtt{reconstruct}$ returns $x^{(v)}$. (When $t = \frac{n}{2} - 1$, $n - t$ is a majority.)*

More formally, a one-or-nothing secret sharing scheme is correct *if for any security parameter $\lambda \in \mathbb{N}$, any vector of secrets $(x^{(1)}, \dots, x^{(l)})$, any index $v \in [l]$*

and any subset $S \subseteq [n], |S| \geq n - t$,

$$\Pr \left[x = x^{(v)} : \begin{array}{c} \mathrm{sk}_i \leftarrow \mathtt{setup}(1^\lambda) \textit{ for } i \in [n] \\ s \leftarrow \mathtt{share}(\mathrm{sk}_1, \ldots, \mathrm{sk}_n, x^{(1)}, \ldots, x^{(l)}) \\ \overline{s}_i \leftarrow \mathtt{vote}(\mathrm{sk}_i, v) \textit{ for } i \in S \\ \overline{s}_i \leftarrow \mathtt{vote}(\mathrm{sk}_i, \bot) \textit{ for } i \in [n] \setminus S \\ x \leftarrow \mathtt{reconstruct}(s, \overline{s}_1, \ldots, \overline{s}_n) \end{array} \right] \geq 1 - negl(\lambda),$$

where the probability is taken over the random coins of the algorithms.

Definition 9 (One-or-Nothing Secret Sharing: Privacy). *Informally, this property requires that when no honest parties produce their ballot using v, then the adversary learns nothing about $x^{(v)}$.*

More formally, a one-or-nothing secret sharing scheme is private *if for any security parameter $\lambda \in \mathbb{N}$, for every PPT adversary $\mathcal{A}$, it holds that*

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

in the following experiment:
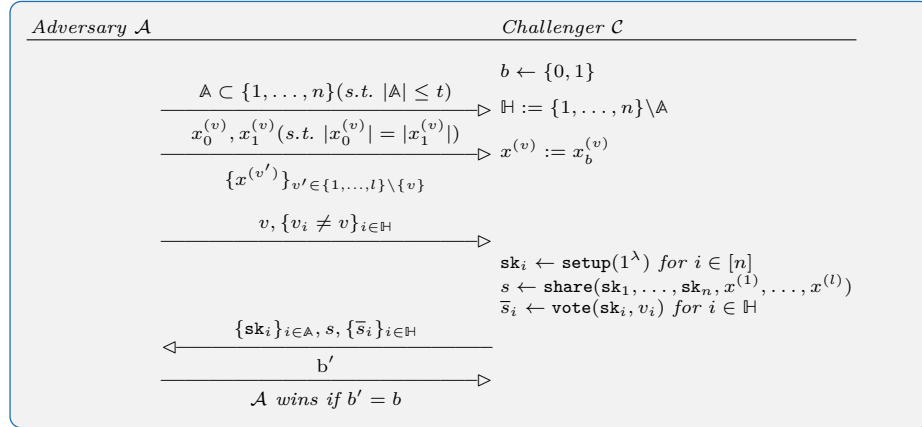
Definition 10 (One-or-Nothing Secret Sharing: Contradiction-Privacy).
Informally, this property requires that if two different parties produce their ballots using different votes $v_i \neq v_j$ such that $v_i \neq \bot$ and $v_j \neq \bot$, then the adversary should learn nothing at all.

More formally, a one-or-nothing secret sharing scheme is contradiction-private *if for any security parameter $\lambda \in \mathbb{N}$, for every PPT adversary $\mathcal{A}$, it holds that*

$$\Pr[\mathcal{A} \textit{ wins}] \leq \frac{1}{2} + negl(\lambda)$$
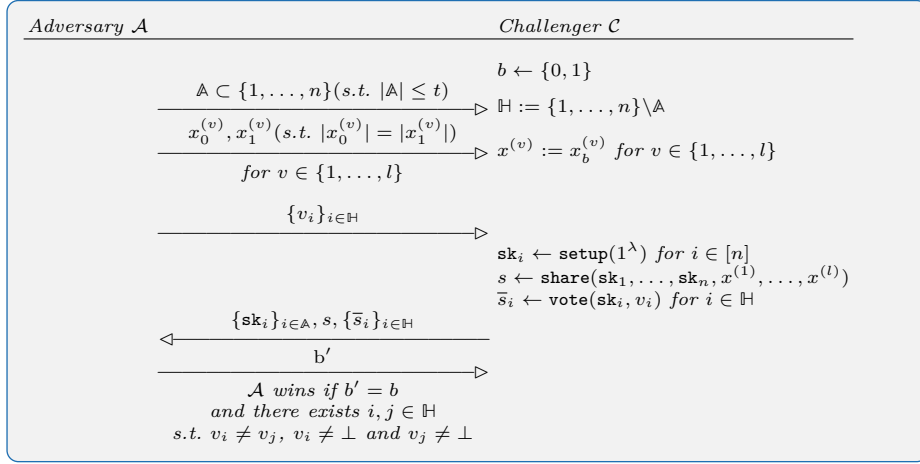
in the following experiment:

24

Adversary $\mathcal{A}$ ... **Challenger $\mathcal{C}$**

$b \leftarrow \{0,1\}$

$\xrightarrow{\quad \mathbb{A} \subset \{1,\dots,n\}\,(s.t.\ |\mathbb{A}| \le t) \quad} \rhd \mathbb{H} := \{1,\dots,n\} \backslash \mathbb{A}$

$\xrightarrow{\quad x_0^{(v)}, x_1^{(v)}\,(s.t.\ |x_0^{(v)}| = |x_1^{(v)}|) \quad} \rhd x^{(v)} := x_b^{(v)} \ for\ v \in \{1,\dots,l\}$

$for\ v \in \{1,\dots,l\}$

$\xrightarrow{\quad \{v_i\}_{i \in \mathbb{H}} \quad}$

$\mathtt{sk}_i \leftarrow \mathtt{setup}(1^\lambda)\ for\ i \in [n]$
$s \leftarrow \mathtt{share}(\mathtt{sk}_1, \dots, \mathtt{sk}_n, x^{(1)}, \dots, x^{(l)})$
$\overline{s}_i \leftarrow \mathtt{vote}(\mathtt{sk}_i, v_i)\ for\ i \in \mathbb{H}$

$\xleftarrow{\quad \{\mathtt{sk}_i\}_{i \in \mathbb{A}}, s, \{\overline{s}_i\}_{i \in \mathbb{H}} \quad}$

$\xrightarrow{\quad b' \quad}$

$\mathcal{A}\ wins\ if\ b' = b$
$and\ there\ exists\ i,j \in \mathbb{H}$
$s.t.\ v_i \ne v_j,\ v_i \ne \bot\ and\ v_j \ne \bot$

7.2 Constructions

A first attempt would be to additively share all the values $x^{(1)}, \dots, x^{(l)}$. However, this fails because if all of the honest parties compute $\mathtt{vote}$ on $\bot$ (by e.g. publishing both their additive shares), the adversary will be able to reconstruct all of the values, violating privacy (Definition 9).

Instead, we instantiate a non-interactive one-or-nothing secret sharing scheme as follows, using a symmetric encryption scheme $\mathtt{SKE} = (\mathtt{keygen}, \mathtt{enc}, \mathtt{dec})$ (defined in the full version of this paper [DMR$^+$20]).

Figure 7.1: Non-Interactive One-or-Nothing Secret Sharing

$\mathtt{setup}(1^\lambda) \to \mathtt{sk}$: Choose $l+1$ symmetric encryption keys $\mathtt{k}^{(1)}, \dots, \mathtt{k}^{(l)}, \mathtt{k}^{(\bot)}$ using $\mathtt{SKE}.\mathtt{keygen}(1^\lambda)$. Let $\mathtt{sk} = (\mathtt{k}^{(1)}, \dots, \mathtt{k}^{(l)}, \mathtt{k}^{(\bot)})$.

$\mathtt{share}(\mathtt{sk}_1, \dots, \mathtt{sk}_n, x^{(1)}, \dots, x^{(l)}) \to s$:
1. Compute $(x_1^{(v)}, \dots, x_n^{(v)})$ as the additive sharing of $x^{(v)}$ for $v \in [l]$.
2. Compute $(x_{i \to 1}^{(v)}, \dots, x_{i \to n}^{(v)})$ as the threshold sharing of $x_i^{(v)}$ with threshold t for $v \in [l]$, $i \in [n]$.
3. Parse $(\mathtt{k}_i^{(1)}, \dots, \mathtt{k}_i^{(l)}, \mathtt{k}_i^{(\bot)}) = \mathtt{sk}_i$ for $i \in [n]$.
4. Compute $c_i^{(v)} = \mathtt{enc}(\mathtt{k}_i^{(v)}, x_i^{(v)})$ for $v \in [l]$, $i \in [n]$.
5. Compute $c_{i \to j}^{(v)} = \mathtt{enc}\big(\mathtt{k}_i^{(\bot)}, \mathtt{enc}(\mathtt{k}_j^{(v)}, x_{i \to j}^{(v)})\big)$ for $v \in [l]$, $i \in [n]$, $j \in [n]$.
6. Output $s = (\{c_i^{(v)}\}_{i \in [n], v \in [l]}, \{c_{i \to j}^{(v)}\}_{i,j \in [n], v \in [l]})$.

$\mathtt{vote}(\mathtt{sk}_i, v) \to \overline{s}_i$ where $v \in \{1, \dots, l, \bot\}$: Output $\overline{s}_i = (v, \mathtt{k}_i^{(v)})$.

$\mathtt{reconstruct}(s, \overline{s}_1, \dots, \overline{s}_n) \to \{x^{(v)}, \bot\}$:
1. Parse $(\{c_i^{(v)}\}_{i \in [n], v \in [l]}, \{c_{i \to j}^{(v)}\}_{i,j \in [n], v \in [l]}) = s$.
2. Parse $(v_i, \mathtt{k}_i) = \overline{s}_i$ for $i \in [n]$.
3. If there does not exist a $v \in \{1, \dots, l\}$ such that at least $(n-t)$ parties vote for v and everyone else votes for $\bot$, output $\bot$.
4. Let $v \ne \bot$ denote the only value which received votes; let $S \subseteq \{1, \dots, n\}$ be the set of i such that $v_i = v$.

5. For $i \in S$ (so, $v_i = v$), compute $x_i = \texttt{dec}(\mathtt{k}_i, c_i^{(v)})$.
6. For $i \notin S$ (so, $v_i = \perp$), for each $j \in S$, compute $x_{i \rightarrow j} = \texttt{dec}\big(\mathtt{k}_i, \texttt{dec}(\mathtt{k}_j, c_{i \rightarrow j}^{(v)})\big)$. Let x_i denote the value reconstructed using the threshold shares $\{x_{i \rightarrow j}\}_{j \in S}$.
7. If there exists any i such that $x_i = \perp$, output $\perp$. Else, output $x = \sum_{i=1}^{n} x_i$.

Theorem 8. *The above construction is a secure non-interactive one-or-nothing sharing scheme when $n > 2t$.*

We defer the proof of security to the full version of this paper [DMR$^+$20].

8 Broadcast in the Second Round: Identifiable Abort

In this section, we show a protocol achieving secure computation with identifiable abort in two rounds, with the first round only using peer-to-peer channels, when $t < \frac{n}{2}$.

One could hope that executing a protocol Π_{bc} that requires two rounds of broadcast over one round of peer-to-peer channels followed by one round of broadcast will simply work, just like in the case of one round of broadcast followed by one round of peer-to-peer channels (Section 6). However, this is not the case. When the first round is over peer-to-peer channels, the danger is that corrupt parties might send inconsistent messages to honest parties in that round. Allowing honest parties to compute their second-round messages based on inconsistent first-round messages might violate security. So, we must somehow guarantee that all honest-party second-round messages are based on the same set of first-round messages.

Our protocol follows the structure of the protocols described by Cohen *et al.* [CGZ20]. It is described as a compiler that takes a protocol Π_{bc} which achieves the desired guarantees given two rounds of broadcast, and achieves those same guarantees in the broadcast pattern we are interested in, which has broadcast available in the second round only. In the compiler of Cohen *et al.*, to ensure that honest parties base their second-round messages on the same view of the first round, parties garble and broadcast their second-message functions. In more detail, in the first round the parties secret share all the labels for their garbled circuit using additive secret sharing, and send their first-round message from the underlying protocol to each of their peers. In the second round (over broadcast), each party sends their garbled second-message function, and for each bit of first-round message she receives, she forwards her share of the corresponding label in everyone else's garbled circuit. The labels corresponding to *the same set of first-round messages* are reconstructed for each party's garbled second-message function, thus guaranteeing consistency.

We use a similar approach. However, as mentioned in the introduction, there are other challenges to address when our goal is identifiable (as opposed to unanimous) abort. In the techniques of Cohen *et al.*, in the second round, for

each bit of every first-round message, every party P_i must forward to everyone else exactly one of a pair of shares of labels which P_i should have obtained from every other party P_j. However, since the first round is over peer-to-peer channels, P_i can claim that it didn't get the shares of labels from P_j, and the computation must still complete (i.e. the correct label needs to be reconstructed), since it is unclear who to blame — P_i or P_j [8]

An alternative approach might be to use *threshold* secret sharing instead of additive secret sharing to share the garbled labels. In order to ensure that honest parties can either identify a cheater or reconstruct at least one of each pair of labels, we would need to set our secret sharing threshold to be at most $n - t$. However, when $t = \frac{n}{2} - 1$, the adversary only needs one additional honest party's share to reconstruct any given label. If she sends different first-round messages to different honest parties, they will contribute shares of different labels, enabling the adversary to reconstruct both labels for some input wires. This allows the adversary to violate honest parties' privacy.

This is where our *non-interactive one-or-nothing secret sharing* primitive comes into play. Parties can use it to secret share the pair of labels for each wire of their garbled circuit *by only broadcasting one value — the public share — in the second round.* By the non-interactive design of the one-or-nothing secret sharing scheme, parties don't even need to have seen the public share to contribute to reconstruction, so no party can claim to be unable to contribute. The privacy properties of the scheme guarantee that at most one label per wire will be recovered. Moreover, if an honest party is not sure which label share to choose (which may happen if she did not get a valid first-round message of Π_{bc}), she can still enable the recovery of the appropriate label (by contributing an equivocation ballot).

We also have to consider how to identify an adversary that sends different first-round messages *from the underlying protocol* to different honest parties. We thus require each party P_i to sign these first-round messages; each other party P_j will only act upon first-round messages from P_i with valid signatures, and echo those messages (and signatures). In this way, we can identify P_i as a cheater as long as she included valid signatures with her inconsistent messages. If she did not, then either enough parties will complain about P_i to implicate her, or the equivocation ballots will allow the computation to complete anyway.

At a very high level, our protocol can be described as follows. In the first round, the parties send their first-round message of Π_{bc} along with a signature to each of their peers. In the second round (over broadcast), the parties do the following: (1) compute a garbling of their second-message function; (2) secret share all the labels for their garbled circuit using the one-or-nothing secret sharing; (3) vote for the share of the corresponding label (based on the first-round message received) in everyone else's garbled circuit; (4) compute a zero-knowledge proof to ensure the correctness of the actions taken in the second round; and (5) echo

---

[8] Note that this is not an issue in the protocol with unanimous abort of Cohen *et al.* since if the reconstruction of the label fails, the honest parties can simply abort.

all the first-round messages of $\Pi_{\mathtt{bc}}$ with the corresponding signatures received from the other parties in the first round.

Intuitively, our protocol achieves identifiable abort due to the following. First, if a corrupt party is not caught, she must have sent a first-round message with a valid signature to at least one honest party; otherwise, $n - t > t$ parties would claim to have a conflict with her, which implicates her as a cheater (since at least one honest party is clearly accusing her). Second, she must *not* have sent two different first-round messages with valid signatures; otherwise, those two contradictory signatures would implicate her. Third, the zero-knowledge proof in the second round ensures that every corrupt party garbles and shares its garbled circuit labels correctly. We can conclude that, by the correctness property of the secret sharing scheme, if no party is caught, then one label from each label pair is reconstructed, and the underlying protocol $\Pi_{\mathtt{bc}}$ can be carried out.

We state the theorem below, and defer the formal description of the protocol to the full version of this paper [DMR+20].

Theorem 9 (P2P-BC, ID, $n > 2t$). *Let $\mathcal{F}$ be an efficiently computable n-party function and let $n > 2t$. Let $\Pi_{\mathtt{bc}}$ be a two broadcast-round protocol that securely computes $\mathcal{F}$ with identifiable abort with the additional constraint that the straight-line simulator can extract inputs from the first-round messages. Assuming a setup with CRS and PKI, and that $(\mathtt{garble}, \mathtt{eval}, \mathtt{simGC})$ is a secure garbling scheme, $(\mathtt{gen}, \mathtt{sign}, \mathtt{ver})$ is a digital signature scheme, $(\mathtt{share}, \mathtt{vote}, \mathtt{reconstruct}, \mathtt{verify})$ is a one-or-nothing secret sharing scheme, $(\mathtt{keygen}, \mathtt{keyagree})$ is a non-interactive key agreement scheme and $(\mathtt{setupZK}, \mathtt{prove}, \mathtt{verify}, \mathtt{simP}, \mathtt{simP.Extract})$ is a secure non-interactive zero-knowledge proof system. Then, there exists a protocol that securely computes $\mathcal{F}$ with identifiable abort over two rounds, the first of which is over peer-to-peer channels, and the second of which is over a broadcast channel.*

Remark 2. Note that when the underlying protocol $\Pi_{\mathtt{bc}}$ is instantiated using the protocols of Gordon *et al.* or Cohen *et al.* [GLS15,CGZ20], then our construction relies only on CRS and PKI (and does not require correlated randomness).

References

ACGJ18. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In *CRYPTO 2018, Part II*, LNCS. Springer, Heidelberg, August 2018.

ACGJ19. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In *EUROCRYPT 2019, Part II*, LNCS. Springer, Heidelberg, May 2019.

BGI+01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO 2001*, LNCS. Springer, Heidelberg, August 2001.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*. ACM Press, May 1988.

BMMR21. Saikrishna Badrinarayanan, Peihan Miao, Pratyay Mukherjee, and Divya Ravi. On the round complexity of fully secure solitary mpc with honest majority. Cryptology ePrint Archive, Report 2021/241, 2021. `https://eprint.iacr.org/2021/241`.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*. ACM Press, May 1988.

CD01. Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In *CRYPTO 2001*, LNCS. Springer, Heidelberg, August 2001.

CGZ20. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In *EUROCRYPT 2020, Part II*, LNCS. Springer, Heidelberg, May 2020.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*. ACM Press, May 1986.

DMR$^+$20. Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Broadcast-optimal two round mpc with an honest majority. Cryptology ePrint Archive, Report 2020/1254, 2020. `https://eprint.iacr.org/2020/1254`.

DS83. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

FGMv02. Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *EUROCRYPT 2002*, LNCS. Springer, Heidelberg, April / May 2002.

FL82. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

GIKR01. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM STOC*. ACM Press, July 2001.

GIKR02. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. On 2-round secure multiparty computation. In *CRYPTO 2002*, LNCS. Springer, Heidelberg, August 2002.

GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO 2015, Part II*, LNCS. Springer, Heidelberg, 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 *19th ACM STOC*. ACM Press, May 1987.

GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT 2018, Part II*, LNCS. Springer, Heidelberg, April / May 2018.

IK02. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP 2002*, LNCS. Springer, Heidelberg, July 2002.

IKKP15. Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO 2015, Part II*, LNCS. Springer, Heidelberg, August 2015.

IKP10.   Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO 2010*, LNCS. Springer, Heidelberg, August 2010.
Lin01.   Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO 2001*, LNCS. Springer, Heidelberg, August 2001.
PR18.   Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In *CRYPTO 2018, Part II*, LNCS. Springer, Heidelberg, August 2018.
Yao86.   Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*. IEEE Computer Society Press, October 1986.