Minimizing Setup in Broadcast-Optimal Two Round MPC

Ivan Damgård[1], Divya Ravi[1], Luisa Siniscalchi[2(✉)], and Sophia Yakoubov[1]

[1] Aarhus University, Aarhus, Denmark
{ivan,divya,sophia.yakoubov}@cs.au.dk
[2] Technical University of Denmark, Kongens Lyngby, Denmark
luisi@dtu.dk

Abstract. In this paper we consider two-round secure computation protocols which use different communication channels in different rounds: namely, protocols where broadcast is available in neither round, both rounds, only the first round, or only the second round. The prior works of Cohen, Garay and Zikas (Eurocrypt 2020) and Damgård, Magri, Ravi, Siniscalchi and Yakoubov (Crypto 2021) give tight characterizations of which security guarantees are achievable for various thresholds in each communication structure .

In this work, we introduce a new security notion, namely, *selective identifiable abort*, which guarantees that every honest party either obtains the output, or aborts identifying one corrupt party (where honest parties may potentially identify different corrupted parties). We investigate what broadcast patterns in two-round MPC allow achieving this guarantee across various settings (such as with or without PKI, with or without an honest majority).

Further, we determine what is possible in the honest majority setting *without* a PKI, closing a question left open by Damgård *et al.* We show that without a PKI, having an honest majority does not make it possible to achieve stronger security guarantees compared to the dishonest majority setting. However, if two-thirds of the parties are guaranteed to be honest, *identifiable abort* is additionally achievable using broadcast only in the second round.

We use fundamentally different techniques from the previous works to avoid relying on private communication in the first round when a PKI is not available, since assuming such private channels without the availability of public encryption keys is unrealistic. We also show that, somewhat surprisingly, the availability of private channels in the first round does not enable stronger security guarantees unless the corruption threshold is one.

Keywords: Secure computation · Round complexity · Minimal setup

S. Yakoubov—Funded in part by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC).

C. Hazay and M. Stam (Eds.): EUROCRYPT 2023, LNCS 14005, pp. 129–158, 2023.
https://doi.org/10.1007/978-3-031-30617-4_5

1 Introduction

It is known that secure computation is possible in two rounds (whereas one round is clearly not enough). However, most known two-round protocols either only achieve the weakest security guarantee (selective abort) [1], or achieve the best-possible guarantee in their setting by making use of a broadcast channel in both rounds [4,12,15]. Implementing broadcast via a protocol among the parties makes no sense in this setting, as the resulting protocol would require much more than two rounds. However, broadcast can also be done using physical assumptions or external services such as blockchains. This typically means that broadcast is expensive and/or slow, so it is important to try to minimize the usage of broadcast (while achieving as strong a security guarantee as possible).

Before discussing previous work in this direction and our contribution, we establish some useful terminology.

1.1 Terminology

In this work, we categorize protocols in terms of (a) the kinds of communication required in each round, (b) the security guarantees they achieve, (c) the setup they require, and (d) the corruption threshold t they support. We will use shorthand for all of these classifications to make our discussions less cumbersome.

Communication Structure. We refer to protocols that use two rounds of broadcast as BC-BC; protocols that use broadcast in the first round only as BC-P2P; protocols that use broadcast in the second round only as P2P-BC; and protocols that don't use broadcast at all as P2P-P2P.

Note that, when no PKI is available, it is not realistic to assume *private* channels in the first round since it is unclear how such private channels would be realized in practice without public keys. Therefore, in what follows, "P2P" in the first round refers to *open* peer-to-peer channels which an adversary can listen in on [1] – unless we explicitly state otherwise. We do assume the availability of private channels in the second round, since one can broadcast (or send over peer-to-peer channels) an encryption under a public key received in the first round.

Security Guarantees. There are six notions of security that a secure computation protocol could hope to achieve, described informally below.

Selective Abort (SA): A secure computation protocol achieves *selective abort* if every honest party either obtains the output, or aborts.

Selective Identifiable Abort (SIA): A secure computation protocol achieves *selective identifiable abort* if every honest party either obtains the output, or aborts identifying one corrupt party (where the corrupt party identified by different honest parties may potentially be different).

[1] We do assume that the peer-to-peer channels are authenticated.

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.

All the notions above, except SIA have been studied previously. This new notion that we introduce here is strictly stronger than selective abort and incomparable with unanimous abort but weaker than identifiable abort (which demands that all honest parties must be in agreement). Selective abort is the weakest notion of security, and is implied by all the others; guaranteed output delivery is the strongest, and implies all the others. Notably, fairness and identifiable abort are incomparable; selective identifiable abort and unanimous abort are incomparable as well.

We believe the notion of SIA is an interesting conceptual contribution. Unlike SA security where an honest party who aborts would only know that someone misbehaved but has no idea who, SIA guarantees that the honest party learns the identity of someone, who behaved incorrectly. This gives the honest party an option of taking some action against this (locally identified) corrupt party, such as refusing to collaborate in other contexts, withholding payment etc. One may also consider a setting where misbehavior due to a technical error is much more likely than a malicious attack. This could be the case, for instance, if parties are considered reliable and with a good level of system security, implying that the weak point is rather the software running the protocol. In such a case, SIA can be used to help find where the error was in case of an abort, whereas SA will give you no such help.

Of course, it would be preferable to achieve IA rather than SIA; but this may not be possible in all settings. The partial identifiability offered by SIA can be quite useful in such settings (where SIA is possible and IA is not), since SIA does not allow anyone to accuse anyone, in particular, honest players will never accuse each other. This means that, in an honest majority setting, if a party has a conflict with more than t parties, then that party is definitely corrupt. Therefore, if several secure computations are done in a setting where SIA is possible (and IA is not), we can accumulate the conflicts, and may eventually be able to identify a corrupt party such that everyone agrees (namely if that party has a conflict with more than t parties). This will not completely prevent adversarial behavior, but it may limit the number of honest parties that are forced to abort. Such scenarios highlight why SIA is a useful notion to study.

Setup. The following forms of setup, from strongest to weakest, are commonly considered in the MPC literature:

Correlated randomness (CR), where the parties are given input-independent secrets which may be correlated,

A public key infrastructure (PKI), where each party has an independent honestly generated[2] public-secret key pair where the public key is known to everyone, and

A common reference string (CRS), where no per-party information is available, but a single trusted reference string is given.

We focus primarily on protocols that only use a CRS, which is the weakest form of setup (except for the extreme case of no setup at all). To make our prose more readable, when talking about e.g. a secure computation protocol that achieves security with identifiable abort given a CRS and uses broadcast in the second round only, we will refer to it as a P2P-BC, IA, CRS protocol. If we additionally want to specify the corruption threshold t to be x, we call it a P2P-BC, IA, CRS, $t \leq x$ protocol.

1.2 Prior Work

Cohen, Garay and Zikas [7] initiated the study of two-round secure computation with broadcast available in one, but not both, rounds. They showed that, in the P2P-BC setting, UA is possible even given a dishonest majority, and that it is the strongest achievable guarantee in this setting. They also showed that, in the BC-P2P setting, SA is the strongest achievable security guarantee given a dishonest majority.

The subsequent work by Damgård, Magri, Ravi, Siniscalchi and Yakoubov [9] continued this line of inquiry, focusing on the honest majority setting. They showed that given an honest majority, in the P2P-BC setting IA is achievable (but fairness is not), and in the BC-P2P setting, the strongest security guarantee—GOD—is achievable.

The constructions of Cohen *et al.* do not explicitly use a PKI, but they do rely on private communication in the first round, which in practice requires a PKI, as discussed above. The constructions of Damgård *et al.* rely on a PKI even more heavily. The natural open question therefore is: what can be done assuming no PKI—only a CRS, and no private communication in the first round?

We note that the recent work of Goel, Jain, Prabhakaran and Raghunath [14] considers instead the plain model or the availability only of a bare PKI (where it is assumed that corrupt parties may generate their public key maliciously). They show that in plain model, in the absence of private channels, no secure computation is possible even given an honest majority. Further, given broadcast (in both rounds) IA is impossible in the plain model, while the strongest guarantee of GOD is feasible in the bare PKI model. Our model is incomparable to that

[2] Throughout this paper, we use the term 'PKI' to refer to a 'trusted PKI', where the PKI keys are assumed to be honestly generated for all parties.

of Goel *et al.* since we consider the availability of a CRS, and communication patterns where broadcast is limited to one of the two rounds.

To the best of our knowledge, the notion of selective identifiable abort is not discussed in previous work.

1.3 Our Contributions

We summarize the contributions of our work in two broad categories, described below.

1.3.1 Introduction of SIA

We introduce and formalize a new security notion of MPC protocols, that we refer to as selective identifiable abort (SIA). Further, we investigate the feasibility of two-round SIA MPC protocols with different broadcast patterns for various settings – with or without PKI, and with or without honest majority. As it turns out, SIA is an interesting notion because it can be achieved in cases where previously only weaker or incomparable notions were known to be possible. Notably, for the P2P-P2P or BC-P2P and $t < n/3$ settings, SIA can be achieved and is the best possible guarantee, where previously only selective abort was known. Note that, with only selective abort, stopping honest players from getting the output is basically without consequences for the adversary, while with SIA, each honest player will identify at least one corrupt player.

In the following we explain all the results on SIA in more detail: In Theorem 3, we show that any BC-BC (respectively P2P-BC) protocol (with some additional properties) can be transformed to an SIA protocol for the same corruption threshold where the second round communication is over peer-to-peer channels. Plugging in the appropriate IA protocols to this theorem yields several positive results, summarized in Table 1 (for the CRS setting) and Sect. 1.4.3 (for the PKI setting). Namely, we obtain that when we assume only a CRS and no private communication in the first round, SIA is achievable in the P2P-P2P setting with $t < \frac{n}{3}$ and in the BC-P2P setting with $t < n$; finally when a PKI is available, SIA is also possible in the P2P-P2P setting with $t < \frac{n}{2}$.

In light of the above, what remains to be investigated are patterns where broadcast is not available in the first-round, for settings with only a CRS and honest majority ($\frac{n}{2} > t \geq \frac{n}{3}$); and settings with PKI and dishonest majority ($\frac{n}{2} \leq t < n$).

In the CRS only setting, we show that P2P-BC, SIA is impossible to achieve even with an honest majority (Theorem 4, this result holds even when the first-round communication is private). Finally, we observe that the impossibility of P2P-BC, IA, PKI protocols for $t < n$ in [7] can be extended to SIA as well (elaborated in the full version [10]).

1.3.2 Complete Characterization of Two-Round MPC in the CRS Model with Honest Majority

Assuming only a CRS and no private communication in the first round, we give a complete characterization of what can be done in two rounds with respect to all the other security guarantees and different broadcast patterns.

In a nutshell, we show that assuming only a CRS, an honest majority does not give much of an advantage over a dishonest majority: regardless of the corruption threshold, IA continues to remain impossible in the P2P-BC setting (directly follows from impossibility of SIA in this setting, Theorem 4) and in the BC-P2P setting, UA continues to remain impossible (Theorem 5)[3]. The latter extends the impossibility result of Patra and Ravi [20], which holds for $n \leq 3t$ (but does not hold for $t > 1$ and any n).

However, if at least two thirds of the parties are honest, in the P2P-BC setting IA is additionally possible (Theorem 2). To show this we give a construction based on a new primitive called *one-or-nothing secret sharing with intermediaries* (adapted from one-or-nothing secret sharing [9]), which may be of independent interest.

Most of our lower bounds hold even given private communication in the first round; however, our constructions do not require it. This shows that surprisingly, in most cases, having private communication in the first round cannot help achieve stronger guarantees.

The one exception is the case where the adversary can only corrupt one party (that is, $t = 1$); for $t = 1$ and $n \geq 4$, guaranteed output delivery can be achieved given private channels in the first round [17,18] even when broadcast is completely unavailable. However, we show that without private channels in the first round fairness (and thus also guaranteed output delivery) is unachievable, even if broadcast is available in both rounds and the adversary corrupts only one participant[4]. We also show that without private channels in the first round, if broadcast is unavailable in the second round, unanimous abort is unachievable.

Finally, we make a relatively simple observation, showing that the positive results from Cohen *et al.* still hold, even without private communication in the first round.

We summarize our findings in Table 1, and the special case of $t = 1$ in Table 2.

1.4 Technical Overview

In Sect. 1.4.1, we summarize our lower bounds; in Sect. 1.4.2, we summarize our constructions. These results assume a setup with CRS only. Lastly, in Sect. 1.4.3, we summarize the results related to SIA, when PKI is available.

[3] Given an additional round of communication instead of a PKI, things look different; Badrinarayanan *et al.* [2] study broadcast-optimal *three*-round MPC with GOD given an honest majority and CRS, and show that GOD is achievable in the BC-BC-P2P setting.

[4] This strengthens the fairness impossibility result of Gordon *et al.* [15] which holds for $n \leq 3t$.

Table 1. Feasibility and impossibility for two-round MPC with different guarantees and broadcast patterns when only a CRS is available (but no PKI or correlated randomness). The R1 column describes whether broadcast is available in round 1; the R2 column describes whether broadcast is available in round 2. In our constructions, round 1 communications are not private; negative results hold even with private round 1 communications. 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. Beige table cells are lower bounds; green table cells are upper bounds.

Without PKI, in Two Rounds

Broadcast Pattern R1	Broadcast Pattern R2	t	Selective Abort (SA)	Selective Identifiable Abort (SIA)	Unanimous Abort (UA)	Identifiable Abort (IA)	Fairness (FAIR)	Guaranteed Output Delivery (GOD)
BC	BC	$\frac{n}{2} \le t < n$	✓	✓	✓	✓[7] w.m.c	✗[6]	✗
P2P	BC		✓	✗[7] (see [10] for details)	✓[7] w.m.c	✗	✗	✗
BC	P2P		✓	✓[7] w.m.c, last round P2P (Theorem 3)	✗[7]	✗	✗	✗
P2P	P2P		✓[7] w.m.c	✗	✗	✗	✗	✗
BC	BC	$\frac{n}{3} \le t < \frac{n}{2}$	✓	✓	✓	✓[7] w.m.c	✗[15, 20]	✗
P2P	BC		✓	✗Theorem 4	✓[7] w.m.c	✗	✗	✗
BC	P2P		✓	✓[7] w.m.c, last round P2P (Theorem 3)	✗[20]	✗	✗[15, 20]	✗
P2P	P2P		✓[7] w.m.c	✗	✗	✗[8]	✗	✗[19]
BC	BC	$t < \frac{n}{3}$	✓	✓	✓	✓[7] w.m.c	✗ for $t > 1$ [13]	✗ for $t > 1$
P2P	BC		✓	✓	✓[7] w.m.c	✓ Theorem 2	✗ for $t > 1$	✗ for $t > 1$
BC	P2P		✓	✓[7] w.m.c, last round P2P (Theorem 3)	✗ for $t > 1$ (Theorem 5)	✗ for $t > 1$	✗ for $t > 1$ [13]	✗ for $t > 1$
P2P	P2P		✓[7] w.m.c	✓ Theorem 2, last round P2P (Theorem 3)	✗ for $t > 1$ [9]	✗ for $t > 1$	✗ for $t > 1$	✗ for $t > 1$

Table 2. Feasibility and impossibility for two-round MPC with different guarantees and broadcast patterns when only a CRS is available, when $t = 1$. We refer to Table 1 for the cases already covered therein.

The $t = 1$ Case

Broadcast Pattern R1	Broadcast Pattern R2	t	Selective Abort (SA)	Selective Identifiable Abort (SIA)	Unanimous Abort (UA)	Identifiable Abort (IA)	Fairness (FAIR)	Guaranteed Output Delivery (GOD)
Without Private Channels in Round 1:								
BC	BC	$t = 1, n > 1$	Table 1	Table 1	Table 1	Table 1	✗Cor 2 [10] ⟶ ✗	
P2P	BC		Table 1	Table 1	Table 1	Table 1	✗ ⟶ ✗	
BC	P2P		Table 1	Table 1	✗Cor 1 [10] ⟶ ✗		✗Cor 1, 2 [10] ⟶ ✗	
P2P	P2P		Table 1	Table 1	✗ ⟶ ✗		✗ ⟶ ✗	
With Private Channels in Round 1:								
Any		$t = 1, n = 4$	✓ ⟵ ✓		✓ ⟵ ✓[17]		✓ ⟵ ✓[17]	
		$t = 1, n \geq 5$	✓ ⟵ ✓		✓ ⟵ ✓[18]		✓ ⟵ ✓[18]	

1.4.1 Lower Bounds

We present several lower bounds, some of which hold even when private channels are available in the first round. This is in contrast to our constructions which avoid the use of private channels before the parties had a chance to exchange public keys.

With Private Channels. In Sect. 4, we present two main lower bounds that hold even if private channels are available in the first round.

Our first lower bound (Theorem 4) shows that P2P-BC, SIA, CRS protocol is impossible when $n \leq 3t$. To show this, we consider a hypothetical 3-party P2P-BC, SIA, CRS protocol where an adversary who controls just one party, say P behaves inconsistently over the first-round peer-to-peer channels and then chooses to act in the second round based on the information sent to one of the honest parties, say P'. Then, SIA guarantees that P' must compute the output even though she finds the pair of remaining parties in conflict, as she cannot decide whom to blame. This makes the protocol vulnerable to an attack by potentially corrupt P' who can simulate this kind of conflict in her head by recomputing the messages of P based on inputs of her choice. Infact, this argument can be extended for $n \leq 3t$.

Our second lower bound (Theorem 5) shows that BC-P2P, UA, CRS protocol is impossible when $t > 1$. To show this, we argue that in any hypothetical BC-P2P, UA, CRS protocol, an adversary who is able to control just two parties is able to perform an even more powerful attack: after execution, she is able to recompute the function output locally on corrupt party inputs of her choice (together with the same fixed set of honest party inputs). This is called a *residual function attack*. This completes the overview of the lower bounds that hold when private channels are present.

When $t = 1$, we show that the availability of private channels makes a difference. When private channels are available in the first round, the strongest guarantee—guaranteed output delivery—is known to be achievable as long as

$n \geq 4$ [17,18]. However, we show in the full version [10] and outline below that without private channels in the first round, the landscape is quite different.

Without Private Channels. In this setting, an adversary can observe all messages sent by an honest party P in the first round; so, those first-round messages cannot suffice to compute the function on P's input—P's second-round messages are crucially necessary for this. If P's first-round messages were enough, the adversary would be able to mount a residual function attack: given P's first-round messages, the adversary would be able to compute the function on P's input (along with inputs of her choice on behalf of the other parties) in her head, by simulating all the other parties. However, if we aim for either unanimous abort (without use of broadcast in the second round) or fairness, we can also argue that P's second-round messages *cannot* be necessary. If we would like to achieve unanimous abort without use of broadcast in the second round, it is important that the adversary not be able to break unanimity by sending different second-round messages to different parties. If we would like to achieve fairness, it is similarly important that the adversary not be able to deny the honest parties access to the output by withholding her second-round messages. So, to achieve either of those goals, the second-round message both must and cannot matter; we thus rule out BC-P2P, UA, CRS protocols (Cor 1 in [10]) and BC-BC, FAIR, CRS protocols (Cor 2 in [10]) when no private channels are available in the first round.

1.4.2 Upper Bounds

Feasibility of P2P-BC, IA, CRS when $t < \frac{n}{3}$ In Sect. 3.2, we present our main positive result, which is a P2P-BC, IA, CRS, $t < \frac{n}{3}$ construction (Fig. 2). Our construction builds on the construction of Damgård *et al.* [9] (which, in turn, builds on the construction of Cohen *et al.* [7]). Like those prior works, we take a protocol that requires two rounds of broadcast, and compile it. Since broadcast is only available in the second round, the key is to ensure that a corrupt party can't break the security of the underlying protocol by sending inconsistent messages to different honest parties in the first round. The solution is to delay computation of the second round messages until parties are sure they agree on what was said in the first round.

Following previous work, we do this by having each party G garble her second-message function (which takes as input all the first-round messages that party expects to receive) and broadcast that garbled circuit in the second round. G additionally secret shares all of the labels for her garbled circuit. We can get identifiable abort from this if we make sure that one of two things happen: (a) sufficiently many parties receive a given first-round message bit coming from a sender S, implying that the label corresponding to that bit is reconstructed (unanimously, over broadcast); or (b) someone is unanimously identified as a cheater. (Of course, two labels for the same input wire should never be reconstructed, since this would compromise the security of the garbled circuit scheme.)

To achieve this, Damgård *et al.* introduce (and use in their construction) the notion of *one-or-nothing secret sharing*. Unfortunately, this primitive crucially relies on a PKI: in the second round, each player must be able to prove that she received a certain message from S in the first round (or abstain if she received nothing). Given a PKI, this can be done by having S sign her first-round messages. Of course, without a PKI, this cannot work as there is no time to agree on public keys.

Therefore, without a PKI, we need a different approach. The approach we use is instead to check in the second round whether there is sufficient consensus among the parties about what S sent in the first round, and only reconstruct the corresponding labels if this is the case. To this end, we define a new primitive in the CRS model called *one-or-nothing secret sharing with intermediaries*. In such a scheme, each garbler G performs two layers of Shamir sharing: first, each label is shared, creating for each party R a share s_{R}. Second, each s_{R} is shared among all parties. Everyone now acts as intermediaries, and passes their sub-shares of s_{R} on to R in the second round. This ensures that a corrupt G cannot fail to deliver a share to R, since G cannot fail to communicate with more than t intermediaries without being identified. Simultaneously, each participant R broadcasts a message enabling the public recovery of only the label share corresponding to what she received from S in the first round. Enough shares for a given label are only recoverable if enough participants received the same bit from S, implicitly implementing the consensus check we mentioned above.

There is one final caveat we need to take care of: the standard network model assumes peer-to-peer "open" channels where the adversary can observe all messages sent. With a PKI, we can make use of private channels (even in the first round), by using public-key encryption (PKE). However, in the absence of a PKI, this makes little sense, so we should *not* use private channels in the first round. Under this constraint we cannot send shares of secrets in the first round. So, we need to figure out a way for G to send sub-shares of R's share s_{R} to intermediaries, and for intermediaries to pass these sub-shares on to R, in a *single* round of broadcast.

The approach we use is as follows: in the second round, for each sub-share of s_{R} intended for intermediary I, G will broadcast an encryption c of that sub-share, under a public key received from I in the first round. Simultaneously, I passes on all of sub-shares to R by broadcasting *transfer keys*. Depending on which value should be decrypted, R broadcasts the relevant decryption key which enables the recovery of the corresponding plaintext. We informally refer to this approach as *transferrable encryption*, where a party is able to transfer decryption capabilities to another, even without first seeing the ciphertext in question.

Our construction of *one-or-nothing secret sharing with intermediaries* relies on a CPA-secure PKE scheme and non-interactive zero-knowledge (NIZK) proof system. This is used as a building block in our P2P-BC, IA, CRS, $3t < n$ construction (formalized in Fig. 2) following the above blueprint.

Modifying Prior P2P-BC Constructions. The work of Cohen *et al.* gives a construction in the P2P-BC and P2P-P2P settings that uses only a CRS (not a

PKI); however, they use private communication in the first round. We observe that we can modify their construction in a straightforward way to only use *public* peer-to-peer communication in the first round, which is more realistic without a PKI. Their construction is a compiler, and in the first round, two things are sent: messages from the underlying construction; and (full-threshold) secret shares of garbled circuit labels, which need to be communicated privately, and which are then selectively published in the second round. Let's pick an underlying construction that uses public communication only (e.g. the construction of [12]). Now, to avoid private communication in the first round, we modify the protocol to delay secret sharing until the second round. Instead, the only additional thing the parties do in the first round is exchange public encryption keys. Like in our construction (described above), it might look like delaying secret sharing poses a problem, since the share recipients need to broadcast the relevant shares to enable output recovery, but if they only receive their shares in the second (last) round, they don't have time to do this. So, we have the share sender G encrypt each share meant for receiver R under a one-time public key belonging to R. Simultaneously, R will publish the corresponding secret key if and only if she wishes to enable the reconstruction of that label.[5] In this way, the same guarantees can be achieved without using private communication in the first round.

Feasibility Results for SIA. In Sect. 3.3, we argue that a BC-BC protocol (respectively a P2P-BC) Π_{bc} that securely computes f with identifiable abort can be turned into an SIA protocol Π (with the same corruption threshold) where the second round is run over peer-to-peer channels, as long as Π_{bc} satisfies the following two properties: 1) the simulator can extract inputs from the first-round; 2) it is efficient to check whether a given second-round of the protocol is correct.

The protocol Π works in the same way as Π_{bc}, except that the second round is sent over peer-to-peer channels. Intuitively, the only advantage that the adversary has in Π is to send inconsistent last round messages. However, we argue that this cannot lead to a pair of honest parties obtaining two different non-$\perp$ outputs. This is because of our assumption that the simulator of Π_{bc} extracts input from the first round messages (and say receives the output y from the ideal functionality). This means that no matter what second round messages the adversary sends in Π_{bc}, the output can never be $y' \neq \perp$ such that $y' \neq y$. More specifically, the adversary's second round messages in Π_{bc} can only determine whether all the honest parties learn y or all the honest parties learn the identity of a cheater (can be potentially chosen by the adversary during the second broadcast round in Π_{bc}, say, by making a corrupt party stop sending messages or sending invalid messages in the second round). Since these second round messages are now sent over peer-to-peer channels instead (but it is possible to efficiently check their validity), we can conclude that each honest party

[5] Note that the full power of our one-or-nothing secret sharing with intermediaries is not necessary here; in our construction, we only require two levels of sharing and intermediaries in order to achieve *identifiable* abort, while this construction aims only for selective and unanimous abort in the two different settings respectively.

in Π would either learn the output y or the identity of a cheater (depending on the version of the second round message the adversary sends privately). It may be the case that honest parties learn different cheaters or some of them learn the output y while others don't; however, this suffices for SIA guarantee.

In Sect. 3.3 we give candidate constructions of BC-BC protocol (respectively a P2P-BC) with identifiable abort, that have the additional properties described above and can thereby be used to yield the BC-P2P (respectively P2P-P2P) SIA upper bounds.

1.4.3 Completing the Picture of SIA with PKI

Given that the notion of selective identifiable abort is introduced in this work, we also investigate how it affects the landscape when a PKI is available. This setting was studied by Damgård *et al.* [9] for the case of honest majority and Cohen *et al.* [7] for the case of dishonest majority.

The case of BC-BC is already settled by Cohen *et al.*, who give an IA construction (stronger than SIA) for $t < n$, relying just on CRS. Next, we note that our observation in Sect. 3.3 lets us transform the above into an SIA protocol (with the same corruption threshold) where the second round is run over peer-to-peer channels; settling the case of BC-P2P setting.

In the P2P-BC setting, we observe that the impossibility of P2P-BC, IA, PKI protocols for $t < n$ in Cohen *et al.* can be extended to SIA as well (see full version [10] for details). However, assuming an honest majority ($t < \frac{n}{2}$), feasibility of SIA follows directly from the P2P-BC, IA, PKI construction of Damgård *et al.*. Applying the observation in Sect. 3.3 to this IA construction of Damgård *et al.*, let us achieve SIA for P2P-P2P setting with the same threshold.

This settles the question of feasibility of two-round SIA with various broadcast patterns in the PKI setting.

1.5 Broadcast Complexity

In the previous two works, no attempt was made to minimize the broadcast overhead of the compilers. They all require the broadcast of garbled second-message functions, the size of which often scales with the complexity of the function computed, which is potentially large. We observe that a generic broadcast optimization (which is folklore, and has appeared in some previous work [5,11,16]) can be applied to any message which is already known to the sender in the first round, but need not be broadcast until the second round. Using this optimization, the size of the additional broadcasts that our compiler—and the compilers of Cohen *et al.* and Damgård *et al.*—becomes independent of the size of the function being computed.

The broadcast optimization is quite straightforward. It enables reliable broadcast of arbitrarily long messages, while only sending fixed-length messages over the broadcast channel in the second round. The dealer sends its message to all the recipients over peer-to-peer channels in the first round. Each recipient then echos the message it received *over peer-to-peer channels* in the second

round. Finally, in the second round, each party also broadcasts a *hash* of the message. If there exists a majority of parties who broadcast the same hash h, then each honest party outputs a pre-image of h. (Each party must have received a pre-image of h because at least one of the broadcasters of h must be honest.) Otherwise, honest parties blame the dealer. Only hashes are sent over the broadcast channel, and the size of those hashes is independent of the size of the message.

Finally, we note that when applying this optimization to our construction, and that of Cohen *et al.* and Damgård *et al.*, garbled circuits which were previously not broadcast until the second round are now sent (over peer-to-peer channels) in the first round. This necessitates the use of adaptive garbled circuits[6].

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 [7]. As in their work, we state our results in a stand-alone setting.[7]

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). 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 peer-to-peer (P2P) network. If peer-to-peer communication occurs in the first round without a PKI, we assume these channels are "open"; that is, the adversary sees all messages sent.[8] In other cases, we assume that these channels can be private, since communications can be encrypted using public keys that are either available via a PKI or exchanged in the first round. In all cases, we assume the channels to be ideally authenticated.

[6] Adaptive garbling schemes [3] remain secure against an adversary who obtains the garbled circuit and then selects the input.

[7] We note that our security proofs can translate to an appropriate (synchronous) composable setting with minimal changes. We also give the formal definition of the new security notion of selective identifiable abort (sl-idabort).

[8] Some of our negative results hold even if private peer-to-peer channels are available in the first round. However, our positive results do not make use of such channels.

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, and the adversary outputs an arbitrary function of the corrupt parties' view. The view of a party during the execution consists of its input, random coins and the messages it sees during the execution.

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 $\mathtt{REAL}_{\Pi,\mathcal{I},\mathcal{A}(\mathsf{aux})}(x, \lambda)$, 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), selective identifiable abort (sl-idabort), unanimous abort (un-abort), identifiable abort (id-abort), fairness (fairness) and guaranteed output delivery (god).

Definition 2 (Ideal Computation). *Consider* type $\in$ {sl-abort, un-abort, sl-idabort,id-abort, fairness, 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 $\mathtt{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathcal{S}(\mathsf{aux})}(x, \lambda)$, 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 $\in$ {sl-abort, un-abort, sl-idabort,id-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 $\in$ {un-abort, fairness}*: The simulator can send* abort *to the trusted party. If it does, we take $\mathcal{J} = [n] \setminus \mathcal{I}$.*
   (c) If type = sl-idabort*: 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.) For each party j in $\mathcal{J}$, the adversary*

selects a corrupt party $i_j^* \in \mathcal{I}$ who will be blamed by party j. It sends $(\texttt{abort}, \mathcal{J}, \{j, i_j^*\}_{j\in\mathcal{J}})$ to the trusted party.

(d) If $\mathsf{type} = \mathsf{id\text{-}abort}$*: If it chooses to abort, the simulator $\mathcal{S}$ can select a corrupt party $i^* \in \mathcal{I}$ who will be blamed, and send $(\texttt{abort}, i^*)$ 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* $\mathsf{type} \in \{\mathsf{sl\text{-}abort}, \mathsf{un\text{-}abort}, \mathsf{fairness}\}$*, we let* $\mathsf{abortmsg} = \perp$.
         – *if* $\mathsf{type} = \mathsf{sl\text{-}idabort}$*, we let* $\mathsf{abortmsg} = \{\mathsf{abortmsg}_j\}_{j\in\mathcal{J}} = (\perp, i_j^*)_{j\in\mathcal{J}}$.
         – *if* $\mathsf{type} = \mathsf{id\text{-}abort}$*, we let* $\mathsf{abortmsg} = (\perp, i^*)$.
      ii. The trusted party sends y_j to every party P_j, $j \in [n] \setminus \mathcal{J}$.
      If $\mathsf{type} = \mathsf{sl\text{-}idabort}$*, the trusted party then sends* $\mathsf{abortmsg}_j$ *to each party P_j, $j \in \mathcal{J}$; otherwise, the trusted party sends* abortmsg *to every party P_j, $j \in \mathcal{J}$*
   Note that, if $\mathsf{type} = \mathsf{god}$*, 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* $\mathsf{type} \in \{\mathsf{sl\text{-}abort}, \mathsf{un\text{-}abort}, \mathsf{sl\text{-}idabort}, \mathsf{id\text{-}abort}, \mathsf{fairness}, \mathsf{god}\}$*. 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 $\mathcal{A}$ with auxiliary input* aux*, there exists a PPT simulator $\mathcal{S}$ such that for every $\mathcal{I} \subseteq [n]$ of size at most t, for all $x \in (\{0,1\}^*)^n$, for all large enough $\lambda \in \mathbb{N}$, it holds that*

$$\texttt{REAL}_{\Pi,\mathcal{I},\mathcal{A}(\mathsf{aux})}(x,\lambda) \stackrel{c}{\equiv} \texttt{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathcal{S}(\mathsf{aux})}(x,\lambda).$$

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: $\texttt{frst-msg}_i$, to compute P_i's first messages to its peers; $\texttt{snd-msg}_i$, to compute P_i's second messages; and $\texttt{output}_i$, to compute P_i's output. Thus, a protocol Π can be defined as $\{(\texttt{frst-msg}_i, \texttt{snd-msg}_i, \texttt{output}_i)\}_{i\in[n]}$.

The syntax of the algorithms is as follows:

– $\texttt{frst-msg}_i(x_i, r_i) \to (\mathsf{msg}^1_{i\to 1}, \ldots, \mathsf{msg}^1_{i\to n})$ 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.

- $\texttt{snd-msg}_i(x_i, r_i, \mathsf{msg}^1_{1\to i}, \ldots, \mathsf{msg}^1_{n\to i}) \to (\mathsf{msg}^2_{i\to 1}, \ldots, \mathsf{msg}^2_{i\to n})$ produces the second-round messages of party P_i to all parties.
- $\texttt{output}_i(x_i, r_i, \mathsf{msg}^1_{1\to i}, \ldots, \mathsf{msg}^1_{n\to i}, \mathsf{msg}^2_{1\to i}, \ldots, \mathsf{msg}^2_{n\to i}) \to y_i$ produces the output returned to party P_i.

We implicitly assume that all of these algorithms also take a CRS as input when one is available.

When the first round is over broadcast channels, we consider $\texttt{frst-msg}_i$ to return only one message—msg^1_i. Similarly, when the second round is over broadcast channels, we consider $\texttt{snd-msg}_i$ to return only msg^2_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 Upper Bounds

We begin with a description of our new primitive, one-or-nothing secret sharing with intermediaries, which is used as a building block in our IA construction. Next, we present our positive results for IA and SIA.

3.1 One-or-Nothing Secret Sharing with Intermediaries

Damgård *et al.* [9] introduce one-or-nothing secret sharing, which allows a dealer to share a vector of secrets in such a way that during reconstruction, at most one of the secrets is recovered (the share holders essentially vote on which one). The correctness guarantee is that if sufficiently many share holders vote for a certain index, and no-one votes against that index (though some parties may equivocate), the value at that index is recovered; the security guarantee is that if at least one party votes for a certain index, the adversary learns nothing about the values at any *other* index. Damgård *et al.* present two versions of this primitive: the default version, and a *non-interactive* version, where parties can vote even if they have not received a share from the dealer. This is done by assuming the dealer shares secret keys with each party, which can be realized via non-interactive key exchange, using a PKI.

Unfortunately, this non-interactive one-or-nothing secret sharing tool (referred to as `1or0`) does not extend to a setting where no PKI is available. In the absence of PKI, the main challenge is to ensure that the share intended for a party, say P, gets delivered (so that her share corresponding to the secret at the index she votes for can be recovered). We achieve this by modeling the fact that other parties can be intermediaries who aid this share transfer. For the setting where only a CRS is available, we propose a new variant of one-or-nothing secret sharing: namely, one-or-nothing secret sharing with intermediaries (referred to as `1or0wi`).

In order to simplify the presentation of our P2P-BC, IA, CRS construction, we define one-or-nothing secret sharing with intermediaries as a *maliciously-secure* primitive. The first round of our protocol is reserved for the exchange

of public keys, so sharing and reconstruction must take place in a single round. The definitions of one-or-nothing secret sharing with intermediaries capture the fact that keys may not have been exchanged consistently, but demand that reconstruction succeeds if blame cannot be assigned. We discuss the syntax, definitions and construction of one-or-nothing secret sharing with intermediaries below.

3.1.1 Syntax

A one-or-nothing secret sharing scheme [9] consists of four algorithms: `setup`, `share`, `vote`, and `reconstruct`. `setup` returns a shared secret key belonging to the dealer and one of the receivers; these keys are then used within `share`, and again in `vote`. To make our one-or-nothing secret sharing with intermediaries secure against malicious adversaries, we move to a public-key syntax, which makes it easier to check parties' behavior using zero knowledge proofs. We change `setup` to return a common reference string crs; keys are then produced by `keygen`, which creates a key pair for one of the receivers. `share`, `vote` and `reconstruct` now all expect the receivers' public keys as input. The syntax of `reconstruct` is modified to support cheater identification; if sufficiently many (at least $n-t$) parties vote for the same value, then either the secret corresponding to this value will be reconstructed, or a cheating party will be identified. We present the syntax of the maliciously-secure one-or-nothing secret sharing with intermediaries below.

$\texttt{setup}(1^\lambda) \rightarrow crs$ is an algorithm which takes as input the security parameter and generates the common reference string.

$\texttt{keygen}(crs) \rightarrow (\texttt{sk}, \texttt{pk})$ is an algorithm which takes as input the common reference string and generates a key pair.

$\texttt{share}(crs, \texttt{pk}_1, \ldots, \texttt{pk}_n, \texttt{z}^{(1)}, \ldots, \texttt{z}^{(l)}) \rightarrow s$ is an algorithm run by the dealer D which takes as input all the parties' public keys, and the l values that are being shared. It outputs a single share s.

$\texttt{vote}(crs, \texttt{sk}_i, \texttt{pk}_1, \ldots, \texttt{pk}_n, v_i) \rightarrow \overline{s}_i$ is an algorithm run by party i which takes as input party i's secret key, all the parties' public keys, and a vote v_i, where $v_i \in \{1, \ldots, l, \perp\}$ 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 returns a ballot $\overline{s}_i$.
Note that, to allow `share` and `vote` to be executed in a single round, `vote` does not take as input the share s.

$\texttt{reconstruct}(crs, s, (\texttt{pk}_1, v_1, \overline{s}_1), \ldots, (\texttt{pk}_n, v_n, \overline{s}_n)) \rightarrow \{\texttt{z}^{(v)}, \perp, \perp_i\}$ is an algorithm which takes as input the output of `share` run by the dealer D, the outputs of `vote` run by each of the n parties, as well as their votes, and outputs the value $\texttt{z}^{(v)}$ which received a majority of votes, *or* $\perp$, *or* $\perp_i$ where i denotes the identity of a cheater.

3.1.2 Security

We require one-or-nothing secret sharing with intermediaries to satisfy *privacy* and *identifiability*, described below. Notice that identifiability naturally

implies correctness. Our definitions of privacy and identifiability both assume that corrupt parties might provide honest parties, including the dealer, with inconsistent or incorrect public keys. Below, we denote the set of n parties as $\{D, P_1, \ldots, P_{n-1}\}$, where D denotes the dealer.

Informal Definition 1 (`1or0wi`: Privacy) *Informally, this property requires that when fewer than $n - 2t$ honest parties produce their ballot using v, then the adversary learns nothing about $\mathtt{z}^{(v)}$.*

The one-or-nothing secret sharing of Damgård *et al.* [9] additionally required *contradiction-privacy*. This guaranteed the privacy of all secrets when a pair of honest parties produce ballots for different indices. Notably, our one-or-nothing secret sharing with intermediaries does not have this property; however, when $n > 3t$, the privacy property implies that at most one secret is reconstructed.[9]

Informal Definition 2 (`1or0wi`: Identifiability) *Informally, this property requires that when at least $n - t$ parties produce their ballot using the same v, either* `reconstruct` *returns $\mathtt{z}^{(v)}$ or a corrupt party is identified.*

It is easy to see that the identifiability property defined above implies *correctness* (i.e. when all algorithms are executed honestly, if at least $n - t$ parties produce their ballot using the same v, `reconstruct` returns $\mathtt{z}^{(v)}$).

We refer to the full version [10] for the formal definitions of privacy and identifiability.

3.1.3 Construction

Both the one-or-nothing secret sharing scheme of Damgård *et al.* [9] and our construction of one-or-nothing secret sharing with intermediaries make use of two layers of Shamir secret sharing. However, Damgård *et al.* crucially differ in the way in which the sub-shares for reconstructing a given value are transferred by the shareholders. Because without a PKI a dealer might not communicate reliably/verifiably to all share recipients (as either she or they might be corrupt), in order to achieve identifiability in such scenarios, we introduce a new tool which we informally call *transferrable encryption*.

Transferrable encryption allows a sender to encrypt a message to an intermediary, who, even before seeing the ciphertext, can *transfer* the ability to decrypt to another receiver. This can be achieved, for instance, simply by having the intermediary encrypt her (single-use) secret decryption key to the receiver.

We now informally describe the one-or-nothing secret sharing with intermediaries algorithms `keygen`, `share`, `vote`, and `reconstruct`:

[9] If we consider the more general case of $t' \leq t$ corruptions, the adversary would learn the secret at an index v only if at least $(n - t - t')$ honest parties vote for v (as these along with the t' ballots known on behalf of the corrupt parties would allow the secret to be reconstructed). Therefore, for the adversary to learn secrets at two different indices, there must exist two disjoint sets of at least $(n - t - t')$. This could happen only if $2(n - t - t') \leq n - t'$, which implies $n \leq 2t + t' \leq 3t$ (as $t' \leq t$); which contradicts our assumption of $n > 3t$.

1. Informally, `keygen` generates many single-use public-key encryption key pairs for each party i, designated for transference of decryption power to different parties j. Each party i will end up with a key pair $(\mathtt{sk}_{j\to i}^{(v)}, \mathtt{pk}_{j\to i}^{(v)})$ for every party j and shared value index v.
2. In the `share` algorithm the dealer threshold secret shares each secret $\mathtt{z}^{(v)}$ as $s_1^{(v)}, \ldots, s_n^{(v)}$, and then threshold secret shares each $s_i^{(v)}$ as $s_{i\to 1}^{(v)}, \ldots, s_{i\to n}^{(v)}$. Then, the dealer broadcasts an encryption of each sub-share $s_{i\to j}^{(v)}$ under a key $\mathtt{pk}_{i\to j}^{(v)}$ belonging to party j; later, during `vote`, party j will act as an intermediary, and forward that share to party i.
3. `vote` is divided into two sub-steps (the first of which is independent of the party's vote):
   (a) Each party j broadcasts *transfer keys* for each index v and each other party i that can be applied to the encryption of $s_{i\to j}^{(v)}$ (under party j's public key $\mathtt{pk}_{i\to j}^{(v)}$) to make it decryptable using party i's secret decryption key $\mathtt{sk}_{i\to i}^{(v)}$. (Such a transfer key can simply be an encryption of $\mathtt{sk}_{i\to j}^{(v)}$ under party i's public key $\mathtt{pk}_{i\to i}^{(v)}$.)
   (b) To vote for the reconstruction of $\mathtt{z}^{(v)}$, each party i broadcasts her relevant secret decryption key $\mathtt{sk}_{i\to i}^{(v)}$.
4. Finally, the `reconstruct` algorithm decrypts all the shares made available through the broadcast of the relevant decryption keys, and reconstructs $\mathtt{z}^{(v)}$ if at least $n - t$ votes supported v; otherwise, a cheating party is identified.

Finally, to achieve security against an active adversary, each party provides a non-interactive zero-knowledge proof (NIZK) to ensure that each step is honestly computed. Therefore, the `setup` algorithm is also tasked with providing the CRSs required for the NIZKs.

More formally, let $\mathtt{PKE} = (\mathtt{keygen}, \mathtt{enc}, \mathtt{dec})$ be a public key encryption scheme with CPA security, and let $\mathtt{NIZK} = (\mathtt{setupZK}, \mathtt{prove}, \mathtt{verify}, \mathtt{simP}, \mathtt{extract})$ be a non-interactive zero-knowledge proof system for the following relations:

$$\mathcal{R}_{\mathtt{keygen}} = \left\{ \begin{array}{l} \phi = \mathtt{pk} \\ \\ w = (\mathtt{sk}, r) \end{array} \middle| (\mathtt{sk}, \mathtt{pk}) \leftarrow \mathtt{PKE.keygen}(1^\lambda; r) \right\},$$

$$\mathcal{R}_{\mathtt{share}} = \left\{ \begin{array}{l} \phi = \{\mathtt{pk}_{i\to j}^{(v)}, c_{i\to j}^{(v)}\}_{v\in[l],i,j\in[n]} \\ \\ w = \left(\begin{array}{l} \{\mathtt{z}^{(v)}, r^{(v)}, \{r_i^{(v)}, \\ \\ \{r_{i\to j}^{(v)}\}_{j\in[n]}\}_{i\in[n]}\}_{v\in[l]} \end{array} \right) \end{array} \middle| \begin{array}{l} \{(s_1^{(v)}, \ldots, s_n^{(v)}) \leftarrow \mathtt{Shamir.share}(\mathtt{z}^{(v)}; r^{(v)})\}_{v\in[l]} \\ \wedge \{(s_{i\to 1}^{(v)}, \ldots, s_{i\to n}^{(v)}) \leftarrow \mathtt{Shamir.share}(s_i^{(v)}; r_i^{(v)})\}_{v\in[l],i\in[n]} \\ \wedge \{c_{i\to j}^{(v)} \leftarrow \mathtt{PKE.enc}(\mathtt{pk}_{i\to j}^{(v)}, s_{i\to j}^{(v)}; r_{i\to j}^{(v)})\}_{v\in[l],i,j\in[n]} \end{array} \right\},$$

$$\mathcal{R}_{\mathtt{vote}} = \left\{ \begin{array}{l} \phi = \left(\begin{array}{l} \{\mathtt{pk}_{j\to j}^{(v)}, \mathtt{pk}_{j\to i}^{(v)}, \mathtt{tk}_{j\to i}^{(v)}\}_{v\in[l],j\in[n]}, \\ v_i, \mathtt{sk}_{i\to i}^{(v_i)} \end{array} \right) \\ w = \left(\{\mathtt{sk}_{j\to i}^{(v)}, \bar{r}_{j\to i}^{(v)}, r_j^{(v)}\}_{v\in[l],j\in[n]} \right) \end{array} \middle| \begin{array}{l} \{(\mathtt{sk}_{j\to i}^{(v)}, \mathtt{pk}_{j\to i}^{(v)}) \leftarrow \mathtt{PKE.keygen}(1^\lambda; \bar{r}_{j\to i}^{(v)})\}_{j\in[n],v\in[l]} \\ \wedge \{\mathtt{tk}_{j\to i}^{(v)} \leftarrow \mathtt{PKE.enc}(\mathtt{pk}_{j\to j}^{(v)}, \mathtt{sk}_{j\to i}^{(v)}; r_j^{(v)})\}_{v\in[l],j\in[n]} \end{array} \right\}.$$

Figure 1 describes our one-or-nothing secret sharing with intermediaries (1or0wi) scheme.

Figure 1: Construction of 1or0wi

$\mathtt{setup}(1^\lambda)$: Set up and output the common reference strings
$crs_{\mathtt{keygen}} \leftarrow \mathtt{setupZK}(1^\lambda, \mathcal{R}_{\mathtt{keygen}})$,
$crs_{\mathtt{share}} \leftarrow \mathtt{setupZK}(1^\lambda, \mathcal{R}_{\mathtt{share}})$, and
$crs_{\mathtt{vote}} \leftarrow \mathtt{setupZK}(1^\lambda, \mathcal{R}_{\mathtt{vote}})$
for the zero knowledge proof system. Return $crs = (crs_{\mathtt{keygen}}, crs_{\mathtt{share}}, crs_{\mathtt{vote}})$.

$\mathtt{keygen}(crs)$, run by party i:

1. For each $j \in [n]$ and $v \in [l]$, $(\mathtt{sk}_{j\to i}^{(v)}, \mathtt{pk}_{j\to i}^{(v)}) \leftarrow \mathtt{PKE.keygen}(1^\lambda; \bar{r}_{j\to i}^{(v)})$.
2. For each $j \in [n]$ and $v \in [l]$, $\pi_{j\to i}^{(v)} \leftarrow \mathtt{NIZK.prove}(crs_{\mathtt{keygen}}, \phi = \mathtt{pk}_{j\to i}^{(v)}, w = (\mathtt{sk}_{j\to i}^{(v)}, \bar{r}_{j\to i}^{(v)}))$.
3. Let $\mathtt{sk}_i = (\{\mathtt{sk}_{j\to i}^{(v)}, \bar{r}_{j\to i}^{(v)}\}_{j\in[n],v\in[l]})$, and $\mathtt{pk}_i = (\{\mathtt{pk}_{j\to i}^{(v)}, \pi_{j\to i}^{(v)}\}_{j\in[n],v\in[l]})$.
4. Output $(\mathtt{sk}_i, \mathtt{pk}_i)$.

$\mathtt{share}(crs, \mathtt{pk}_1, \ldots, \mathtt{pk}_n, \mathtt{z}^{(1)}, \ldots, \mathtt{z}^{(l)})$, run by the dealer D **(where** $\mathtt{pk}_i = \{\mathtt{pk}_{j\to i}^{(v)}, \pi_{j\to i}^{(v)}\}_{j\in[n],v\in[l]}$**):**

1. For each $v \in [l]$, compute $(s_1^{(v)}, \ldots, s_n^{(v)}) \leftarrow \mathtt{Shamir.share}(\mathtt{z}^{(v)}; r^{(v)})$ as the threshold sharing of $\mathtt{z}^{(v)}$ with threshold $(n-t-1)$.
2. For each $i \in [n]$ and $v \in [l]$, compute $(s_{i\to 1}^{(v)}, \ldots, s_{i\to n}^{(v)}) \leftarrow \mathtt{Shamir.share}(s_i^{(v)}; r_i^{(v)})$ as the threshold sharing of $s_i^{(v)}$ with threshold $(n-2t-1)$.
3. For each $i, j \in [n]$ and $v \in [l]$, compute $c_{i\to j}^{(v)} \leftarrow \mathtt{PKE.enc}(\mathtt{pk}_{i\to j}^{(v)}, s_{i\to j}^{(v)}; r_{i\to j}^{(v)})$.
4. Set
   - $\phi_{\mathtt{share}} = (\{\mathtt{pk}_{i\to j}^{(v)}, c_{i\to j}^{(v)}\}_{v\in[l],i,j\in[n]})$ and
   - $w_{\mathtt{share}} = (\{\mathtt{z}^{(v)}, r^{(v)}, \{r_i^{(v)}, \{r_{i\to j}^{(v)}\}_{j\in[n]}\}_{i\in[n]}\}_{v\in[l]})$.

   Compute $\pi_{\mathtt{share}} \leftarrow \mathtt{prove}(crs_{\mathtt{share}}, \phi_{\mathtt{share}}, w_{\mathtt{share}})$.

5. Set $s = (\phi_{\texttt{share}}, \pi_{\texttt{share}})$ and output s.

$\texttt{vote}(crs, \texttt{sk}_i, \texttt{pk}_1, \ldots, \texttt{pk}_n, v_i)$, run by party i **(where** $\texttt{pk}_i = \{\texttt{pk}_{j\to i}^{(v)}, \pi_{j\to i}^{(v)}\}_{j\in[n], v\in[l]}$ **and** $\texttt{sk}_i = \{\texttt{sk}_{j\to i}^{(v)}, \bar{r}_{j\to i}^{(v)}\}_{j\in[n], v\in[l]}$**):**

1. For each $v \in [l]$ and $j \in [n]$, let $\texttt{tk}_{j\to i}^{(v)} \leftarrow \texttt{PKE.enc}(\texttt{pk}_{j\to j}^{(v)}, \texttt{sk}_{j\to i}^{(v)}; r_j^{(v)})$.
2. Set
   - $\phi_{\texttt{vote},i} = (\{\texttt{pk}_{j\to j}^{(v)}, \texttt{pk}_{j\to i}^{(v)}, \texttt{tk}_{j\to i}^{(v)}\}_{v\in[l], j\in[n]}, v_i, \texttt{sk}_{i\to i}^{(v_i)})$ [a]
   - $w_{\texttt{vote},i} = (\{\texttt{sk}_{j\to i}^{(v)}, \bar{r}_{j\to i}^{(v)}, r_j^{(v)}\}_{v\in[l], j\in[n]})$.

   Compute $\pi_{\texttt{vote},i} \leftarrow \texttt{prove}(crs_{\texttt{vote}}, \phi_{\texttt{vote},i}, w_{\texttt{vote},i})$.
3. Set $\bar{s}_i = (\phi_{\texttt{vote},i}, \pi_{\texttt{vote},i})$ and output $\bar{s}_i$.

$\texttt{reconstruct}(crs, s, (\texttt{pk}_1, v_1, \bar{s}_1), \ldots, (\texttt{pk}_n, v_n, \bar{s}_n))$ **(where** $s = (\{\texttt{pk}_{i\to j}^{(v)}, c_{i\to j}^{(v)}\}_{v\in[l], i,j\in[n]}, \pi_{\texttt{share}})$**,** $\texttt{pk}_i = \{\texttt{pk}_{j\to i}^{(v)}, \pi_{j\to i}^{(v)}\}_{j\in[n], v\in[l]}$ **and** $\bar{s}_i = (\phi_{\texttt{vote},i} = (\{\texttt{pk}_{j\to j}^{(v)}, \texttt{pk}_{j\to i}^{(v)}, \texttt{tk}_{j\to i}^{(v)}\}_{v\in[l], j\in[n]}, v_i, \texttt{sk}_{i\to i}^{(v_i)}), \pi_{\texttt{vote},i})$**):**

Identify the winning vote:

1. If there does not exist a $v \in \{1, \ldots, l\}$ such that at least $(n - t)$ parties vote for v, output $\perp$. Let $S_{\texttt{vote}} \subseteq [n]$ be the set of parties i such that $v_i = v$.

Verify the zero knowledge proofs:

2. For $i, j \in [n]$, if $\texttt{NIZK.verify}(crs_{\texttt{keygen}}, \phi = \texttt{pk}_{j\to i}^{(v)}, \pi_{j\to i}^{(v)}) = \texttt{reject}$ (where $\texttt{pk}_{j\to i}^{(v)}$, $\pi_{j\to i}^{(v)}$ are taken from $\texttt{pk}_i$), return $\perp_i$.
3. If $\texttt{NIZK.verify}(crs_{\texttt{share}}, \phi_{\texttt{share}}, \pi_{\texttt{share}}) = \texttt{reject}$ (where $\phi_{\texttt{share}}$, $\pi_{\texttt{share}}$ are taken from s), return $\perp_D$.
4. For $i \in [n]$, if $\texttt{NIZK.verify}(crs_{\texttt{vote}}, \phi_{\texttt{vote},i}, \pi_{\texttt{vote},i}) = \texttt{reject}$ (where $\phi_{\texttt{vote},i}$, $\pi_{\texttt{vote},i}$ are taken from $\bar{s}_i$), return $\perp_i$.

Check the consistency of the share, ballots and keys:

5. For $i \in [n]$, let $S'_i \subseteq [n]$ be the set of parties $j \in [n]$ such that (a) $\texttt{pk}_{i\to i}^{(v)}$ is the same in $\texttt{pk}_i$ and $\bar{s}_j$, and (b) $\texttt{pk}_{j\to j}^{(v)}$ is the same in $\texttt{pk}_j$ and $\bar{s}_i$. If $|S'_i| < n - t$, return $\perp_i$.
6. Let $S_D \subseteq [n]$ be the set of parties i such that $\{\texttt{pk}_{j\to i}^{(v)}\}_{j\in[n]}$ is the same in $\texttt{pk}_i$ and s. If $|S_D| < n - t$, return $\perp_D$.
7. For $i \in S_{\texttt{vote}}$, let $S_i = S'_i \cap S_D$. Note that $|S_i| \geq n - 2t$.
   For $j \in S_i$, we have a ciphertext $\texttt{tk}_{i\to j}^{(v)}$ (contained in $\bar{s}_j$), a secret key $\texttt{sk}_{i\to i}^{(v)}$ (contained in $\bar{s}_i$) and a ciphertext $c_{i\to j}^{(v)}$ (contained in s). Let $\texttt{sk}_{i\to j}^{(v)} \leftarrow \texttt{PKE.dec}(\texttt{sk}_{i\to i}^{(v)}, \texttt{tk}_{i\to j}^{(v)})$. Let $s_{i\to j}^{(v)} \leftarrow \texttt{PKE.dec}(\texttt{sk}_{i\to j}^{(v)}, c_{i\to j}^{(v)})$.
8. For each $i \in S_{\texttt{vote}}$, let $s_i^{(v)} \leftarrow \texttt{Shamir.reconstruct}(\{s_{i\to j}^{(v)}\}_{j\in S_i})$.
9. Output $\texttt{z}^{(v)} \leftarrow \texttt{Shamir.reconstruct}(\{s_i^{(v)}\}_{i\in S_{\texttt{vote}}})$.

Maliciously secure one-or-nothing secret sharing with intermediaries when $n > 3t$.

[a] any string $m^{(\perp)}$ is to be interpreted as $\perp$.

Theorem 1. *The construction in Fig. 1 is a maliciously secure one-or-nothing secret sharing with intermediaries when $n > 3t$ if* PKE *is a public key encryption scheme with CPA security, and* NIZK *is a secure non-interactive zero-knowledge proof system.*

The proof appears in the full version [10]

3.2 IA Feasibility Result: P2P-BC, IA, $3t < n$

Our upper bounds are based on those of Cohen *et al.* [7] and Damgård *et al.* [9]. They take a BC-BC protocol $\Pi_{\mathtt{bc}}$, and *compile* it to the P2P-BC setting. The primary challenge here is making sure that corrupt parties cannot break security by sending different messages to honest parties in the first round. Our compiler makes sure that if corrupt party first-round messages are *consistent enough*, honest party second-round messages are produced on the same set of first-round messages; otherwise, a corrupt party is unanimously identified. To achieve this, we (and the prior works) have each party garble her second-message function, which has her own input hardcoded, and takes as input all the first-round messages she receives. Each party also secret-shares all of the labels for her own garbled circuit. In the second round, over broadcast, parties echo the first-round messages they received, distribute their garbled circuit, and contribute to label reconstruction (for everyone's garbled circuits) corresponding to the first-round messages they received. If there aren't $n-t$ parties who all echo the same first-round message from a given P_i, honest parties abort blaming P_i; if there aren't $n-t$ parties who all contribute valid ballots for P_j's labels, honest parties abort blaming P_j. Note that if an (identifiable) abort happens, reconstruction is allowed to fail.

Using Shamir secret sharing with threshold $s = \frac{3n}{5}$, this leads to a P2P-BC, IA, CRS protocol with $t < \frac{n}{5}$. The reason we have corruption threshold $t = \frac{n}{5}$ and sharing threshold $s = \frac{3n}{5}$ is that we have two constraints:

1. In order to prevent the adversary from learning two labels for the same wire by sending different first-round messages to two subsets of the honest parties, we need $s \geq t + \frac{n-t}{2}$.
2. In order to ensure that even after (a) t parties echo a different message from party m and (b) a different t parties give bad label shares we still have enough shares to reconstruct, we need $s < n - 2t$. (If only t parties have inconsistent claims with the message sender and a different t parties have inconsistent claims with the label share dealer, we have no idea who to blame, so we *have to* reconstruct!)

We get

$$\begin{aligned} t + \frac{n-t}{2} \leq s < n - 2t \\ \Rightarrow \quad t + n < 2n - 4t \\ \Rightarrow \quad 5t < n. \end{aligned}$$

However, $5t < n$ does not match the lower bound from Theorem 4.

To match the lower bound we need a more sophisticated mechanism of sharing such that *all* parties can contribute valid shares of each label, or someone is unanimously identified as a cheater. In Sect. 3.1 we construct exactly such a primitive, which we call one-or-nothing secret sharing with intermediaries (`1or0wi`). Intuitively, our one-or-nothing secret sharing with intermediaries achieves this goal by having each dealer use all of the parties as intermediaries to all share recipients; if sufficiently many intermediaries don't succeed in helping the dealer G give a share to a recipient P, then either the dealer or the recipient can be identified as corrupt, since they are in conflict with more than t intermediaries (we refer to Sect. 3.1 to a more detailed description of how this works).

We are now ready to describe our final protocol with identifiable abort for threshold $3t < n$. In the first round (which is over public peer-to-peer channels), the parties send their first-round messages of Π_{bc} along with the public keys produced by the key generation algorithm of `1or0wi`. In the second round (which is over broadcast), the parties execute the following steps:

1. They compute a garbling of the second-message function of Π_{bc};
2. they use `1or0wi` to share the labels of their garbled circuit;
3. they use `1or0wi` to vote for the labels of the garbled circuits of the other participants based on the first-round messages of Π_{bc} (received in the peer-to-peer round); and
4. they echo the first-round messages of Π_{bc} received in the first round.

Before computing the output, each party P_i performs some validations on the echoed messages. Namely, P_i checks that (a) all the parties generated their ballots for each garbled circuit based on the first-round messages that they echoed, and (b) all the parties have mutual successful communication with at least $n-t$ others in the first round. If there is a party P_j that does not pass these checks, party P_i identifies P_j as a cheater. If all of the parties pass the checks, then party P_i invokes the `reconstruct` algorithm of `1or0wi`. If `reconstruct` blames party P_j, P_i aborts and identifies that party as a cheater. Otherwise, P_i reconstructs labels for all the garbled circuits, uses the garbled circuits to obtain the second-round messages of Π_{bc}, and uses those second-round messages to complete the protocol and obtain the computation output.

Roughly speaking, the identifiable abort property is guaranteed since the one-or-nothing secret sharing with intermediaries is secure against active adversaries. Therefore, if the two validations (a) and (b) succeed, we can rely on the properties of `1or0wi` to guarantee that Π_{bc} is executed or a malicious party is identified.

More formally our protocol is described in Fig. 2 and we assume that the parties have access to the following tools:

Tools.
- A BC-BC, IA, CRS protocol i.e. a two-round broadcast protocol $\Pi_{\mathtt{bc}}$ achieving security with identifiable abort. (This could, for instance, be the protocol described by Cohen *et al.* [7].)
  $\Pi_{\mathtt{bc}}$ is represented by the set of functions $\{\mathtt{frst\text{-}msg}_i, \mathtt{snd\text{-}msg}_i, \mathtt{output}_i\}_{i\in[n]}$.
- A garbling scheme $(\mathtt{garble}, \mathtt{eval}, \mathtt{simGC})$.
- A one-or-nothing secret sharing with intermediaries $\mathtt{1or0wi} = (\mathtt{setup}, \mathtt{keygen}, \mathtt{share}, \mathtt{vote}, \mathtt{reconstruct})$ (defined in Sect. 3.1).

Notation. Let $\mathtt{C}_i(x_i, r_i, \mathsf{msg}_1^1, \ldots, \mathsf{msg}_n^1)$ denote the boolean circuit that takes P_i's input x_i, randomness r_i and the first round messages $\mathsf{msg}_1^1, \ldots, \mathsf{msg}_n^1$, and outputs msg_i^2. For simplicity assume that (x_i, r_i) consists of z bits, and each first round message is ℓ bits long, so each circuit has $\mathsf{L} = z + n \cdot \ell$ input bits. Note that $\mathtt{C}_i$ is public. Let g be the size of a garbled $\mathtt{C}_i$.

Figure 2: $\Pi_{\mathsf{p2pbc}}^{\mathsf{id\text{-}abort}}$ with $n > 3t$

Private input. Every party P_i has a private input $x_i \in \{0,1\}^*$ and randomness $r_i \in \{0,1\}^*$.

Setup.
- CRS setup for one-or-nothing secret sharing with intermediaries: $crs \leftarrow \mathtt{setup}(1^\lambda)$.
- Setup for $\Pi_{\mathtt{bc}}$ (which includes CRS when instantiated using the protocol of [7]). [a]

First Round. Each party P_i does the following:
1. Let $(\mathtt{sk}_i, \mathtt{pk}_i) \leftarrow \mathtt{keygen}(1^\lambda)$, where $\mathtt{pk}_i = \{\mathtt{pk}_i^{(1)} = (\mathtt{pk}_i^{(1,1)}, \ldots, \mathtt{pk}_i^{(1,\mathsf{L})}), \ldots, \mathtt{pk}_i^{(n)} = (\mathtt{pk}_i^{(n,1)}, \ldots, \mathtt{pk}_i^{(n,\mathsf{L})})\}$ is a vector of $n\mathsf{L}$ public keys with the corresponding vector of secret keys $\mathtt{sk}_i = \{\mathtt{sk}_i^{(1)} = (\mathtt{sk}_i^{(1,1)}, \ldots, \mathtt{sk}_i^{(1,\mathsf{L})}), \ldots, \mathtt{sk}_i^{(n)} = (\mathtt{sk}_i^{(n,1)}, \ldots, \mathtt{sk}_i^{(n,\mathsf{L})})\}$
   (We abuse notation slightly by assuming that $\mathtt{keygen}(1^\lambda)$ outputs a vector of public keys and secret keys; we do this for simplicity)
2. Let $\mathsf{msg}_i^1 \leftarrow \mathtt{frst\text{-}msg}_i(x_i, r_i)$ be P_i's first round message in $\Pi_{\mathtt{bc}}$.
3. Send $(\mathtt{pk}_i, \mathsf{msg}_i^1)$ to P_j for $j \in [n]$.

Second Round. Each party P_i does the following:

We specify multiple broadcast messages separately for clarity; however, they are all sent simultaneously as a single round of communication.

1. Let $\mathtt{pk}_{j\to i} = \{\mathtt{pk}_{j\to i}^{(1)}, \ldots, \mathtt{pk}_{j\to i}^{(n)}\}$ denote the $\mathtt{pk}_j$ received privately from P_j $(j \in [n])$, where $\mathtt{pk}_{j\to i}^{(k)} = (\mathtt{pk}_{j\to i}^{(k,1)}, \ldots, \mathtt{pk}_{j\to i}^{(k,\mathsf{L})})$ for $k \in [n]$.
2. Compute $(\mathtt{GC}_i, \boldsymbol{K}_i) \leftarrow \mathtt{garble}(1^\lambda, \mathtt{C}_i; R_i)$, where $\boldsymbol{K}_i = \{K_{i,l}^{(0)}, K_{i,l}^{(1)}\}_{l\in[\mathsf{L}]}$.

3. For every $l \in [z+1, \ldots, \mathsf{L}]$, let $s_{i,l} \leftarrow \mathtt{share}(crs, \mathtt{pk}_{1\to i}^{(i,l)}, \ldots, \mathtt{pk}_{n\to i}^{(i,l)}, K_{i,l}^{(0)}, K_{i,l}^{(1)})$. Broadcast $\{s_{i,l}\}_{l\in[z+1,\ldots,\mathsf{L}]}$.
4. Let $(\nu_{i,z+1}, \ldots, \nu_{i,\mathsf{L}})$ denote the bits comprising $(\mathsf{msg}_{1\to i}^1, \ldots, \mathsf{msg}_{n\to i}^1)$, where $\mathsf{msg}_{j\to i}^1$ refers to msg_j^1 received from P_j in Round 1.
5. For each $k \in [n]$ and $l \in [z+1, \mathsf{L}]$: Compute and broadcast $\overline{s}_{i,l}^{(k)} \leftarrow \mathtt{vote}(crs, \mathtt{sk}_i^{(k,l)}, \mathtt{pk}_{1\to i}^{(k,l)}, \ldots, \mathtt{pk}_{n\to i}^{(k,l)}, \nu_{i,l})$.

Broadcast own garbled circuit:

6. Let $(\nu_{i,1}, \ldots, \nu_{i,z})$ denote the bits corresponding to (x_i, r_i).
7. For $l \in [z]$, let $K_{i,l} = K_{i,l}^{(\nu_{i,l})}$.
8. Broadcast $(\mathtt{GC}_i, \{K_{i,l}\}_{l\in[z]})$.

Echo first-round messages:

9. Broadcast $(\mathsf{msg}_{1\to i}^1, \ldots, \mathsf{msg}_{n\to i}^1)$.

Let $\mathsf{msg}_i^1 = \mathsf{msg}_{i\to i}^1$ denote the party's own first-round message.

Output Computation. Each party P_i does the following:

If there is a party who did not generate ballots for each garbled circuit based on the first-round messages that she echoed, blame that party:

1. For $j \in [n]$: Check if $\{\mathsf{msg}_{k\to j}^1\}_{k\in[n]}$ broadcast by P_j is consistent with $\{\overline{s}_{j,l}^{(k)}\}_{k\in[n],l\in[z+1,\mathsf{L}]}$ [b] . Output $\mathtt{abort}_j$ if the check fails. Else, set $(\nu_{j,z+1}, \ldots, \nu_{j,\mathsf{L}})$ as the bits comprising $(\mathsf{msg}_{1\to j}^1, \ldots, \mathsf{msg}_{n\to j}^1)$.

If there is a party who did not have mutual successful communication with at least $n-t$ others in the first round, blame that party:

2. For $j \in [n]$: If there does not exist a set $|S_j| \geq n-t$ such that, for $k \in S_j$, $\mathsf{msg}_{j\to k}^1 = \mathsf{msg}_j^1$ holds; output $\mathtt{abort}_j$.

Decrypt the shares:

3. For $k \in [n]$ (whose garbled circuit we will now consider):
   (a) For $l \in [z+1, \mathsf{L}]$, compute $K_{k,l} \leftarrow \mathtt{reconstruct}(crs, s_{k,l}, (\mathtt{pk}_1^{(k,l)}, v_{1,l}, \overline{s}_{1,l}^{(k)}), \ldots, (\mathtt{pk}_n^{(k,l)}, v_{n,l}, \overline{s}_{n,l}^{(k,l)})$. If $\mathtt{reconstruct}$ returns $\perp_{\mathtt{id}}$, output $\mathtt{abort}_{\mathtt{id}}$. Else, continue.
   (b) Evaluate $\mathsf{msg}_k^2 \leftarrow \mathtt{eval}(\mathtt{GC}_k, (K_{k,1}, \ldots, K_{k,\mathsf{L}}))$. If the evaluation fails, output $\mathtt{abort}_k$.
4. Output $y \leftarrow \mathtt{output}_i(x_i, r_i, \mathsf{msg}_1^1, \ldots, \mathsf{msg}_n^1, \mathsf{msg}_1^2, \ldots, \mathsf{msg}_n^2)$.

P2P-BC, IA, $t < \frac{n}{3}$ secure computation in the CRS model.

---

a For simplicity (to avoid introducing additional notation), we assume implicitly that the set of functions $\{\mathtt{frst\text{-}msg}_i, \mathtt{snd\text{-}msg}_i, \mathtt{output}_i\}_{i\in[n]}$ of $\Pi_{\mathtt{bc}}$ use the relevant setup information.

b Note that in our construction of one-or-nothing secret sharing with intermediaries, it is possible to retrieve the corresponding vote directly from the ballot $\overline{s}_{j,l}^{(k)}$.

Theorem 2 (P2P-BC, ID, CRS, $n > 3t$). *Let f be an efficiently computable n-party function and let $n > 3t$. Let Π_{bc} be a BC-BC, ID, CRS protocol that securely computes f with the additional constraint that the straight-line simulator can extract inputs from corrupt parties' first-round messages. Assuming that* $(\mathtt{garble}, \mathtt{eval}, \mathtt{simGC})$ *is a secure garbling scheme, and* $(\mathtt{setup}, \mathtt{keygen}, \mathtt{share}, \mathtt{vote}, \mathtt{reconstruct})$ *is a secure one-or-nothing secret sharing with intermediaries. Then, $\Pi^{\mathsf{id\text{-}abort}}_{\mathsf{p2pbc}}$ securely computes 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 and peer-to-peer channels.*

3.3 Feasibility Results for SIA

Our positive results for SIA rely on the following theorem (we defer its proof to the full version [10]).

Theorem 3. *Let Π_{bc} be a BC-BC protocol (respectively a P2P-BC) that securely computes f with identifiable abort security against t corruptions with the additional properties that the simulator can extract inputs from the first-round messages and it is efficient to check whether a given second-round message is correct. Then Π_{bc} securely computes f with selective identifiable-abort security against t corruptions when the second round is run over peer-to-peer channels instead.*

4 Lower Bounds

Our impossibility results for the setting where the first-round is over private peer-to-peer channels appear below. Our impossibility for the setting with public peer-to-peer channels in the first round appear in the full version [10].

Theorem 4 (P2P-BC, SIA, CRS, $n \leq 3t$). *There exist functions f such that no n-party two-round protocol can compute f with selective identifiable abort against $t \geq \frac{n}{3}$ corruptions while making use of broadcast only in the second round (i.e. where the first round is over peer-to-peer channels[10] and second round uses both broadcast and peer-to-peer channels).*

In our proof, we use the function $f_{\mathtt{ot}}$. Let the input of P_1, P_2 be a pair of strings $x_1 = (\mathtt{z}_0, \mathtt{z}_1)$, $x_2 = (\mathtt{z}'_0, \mathtt{z}'_1)$ where $\mathtt{z}_0, \mathtt{z}_1, \mathtt{z}'_0, \mathtt{z}'_1 \in \{0,1\}^\lambda$, and the input of P_n be a choice bit $x_n = c \in \{0,1\}$. The input of other parties is $\perp$ (i.e. $x_i = \perp$ for $i \in [n] \setminus \{1, 2, n\}$). $f_{\mathtt{ot}}$ allows everyone to learn $(\mathtt{z}_c, \mathtt{z}'_c)$.

Proof. We prove Theorem 4 by contradiction. Let Π be an n-party protocol computing $f_{\mathtt{ot}}$ that achieves identifiable abort against $t \geq \frac{n}{3}$ corruptions by using broadcast in the second round only.

For simplicity, we assume $n = 3$ and $t = 1$. We analyze the following scenarios in an execution of Π.

[10] The peer-to-peer channels can be private or "open".

Scenario 1: The adversary does the following on behalf of P_3.

Round 1. Compute and send messages based on input $x_3 = 0$ and $x_3 = 1$ to P_1 and P_2 respectively. (It is possible for the adversary to send inconsistent first-round messages as the first round is communicated over peer-to-peer channels.)

Round 2. Discard the first-round message from P_2 and send messages based on input $x_3 = 0$. In other words, P_3 pretends as if she behaved honestly using input $x_3 = 0$ and did not receive a peer-to-peer message from P_2 in the first round.

Scenario 2: Consider an adversary who corrupts P_2. Suppose the input of honest P_3 is $x_3 = 0$. The adversary behaves as follows on behalf of P_2:

Round 1. Behave honestly as per protocol specifications, except that the peer-to-peer message to P_3 is not sent.

Round 2. Pretend to have received first round messages from P_3 based on $x_3 = 1$. In more detail, the adversary drops the first round peer-to-peer message received from P_3 and replaces it by locally computing P_3's first round message based on input $x_3 = 1$ and some randomness (that the adversary can sample locally on behalf of P_3). Note that the adversary can do this without being caught, due to the absence of PKI or correlated randomness.

Claim. Π is such that P_1 in Scenario 1 learns the output $(\mathsf{z}_0, \mathsf{z}'_0)$ with all but negligible probability.

Proof. First, we observe that the view of honest P_1 in Scenario 1 is distributed identically to her view in Scenario 2. This is because in both scenarios, P_1 observes the following conflict between P_2 and P_3: P_3 claims to have not received the first-round peer-to-peer message from P_2 while P_2 claims to have received first-round peer-to-peer message from P_3 based on $x_3 = 1$. Therefore, to satisfy the guarantees of SIA, it must hold that either P_1 aborts in both scenarios or obtains the output in both scenarios. The former is not possible, since P_1 would identify the same cheater in both scenarios, which means that she would identify an honest party in one of the two scenarios (as the corrupt party is different in the two scenarios). We can thus infer from selective identifiable abort security guarantee of Π that both the above scenarios result in P_1 receiving an output, with all but negligible probability.

The output obtained by P_1 in Scenario 2 must include z_0 as it should be computed with respect to the input $x_3 = 0$ of honest P_3 and input $(\mathsf{z}_0, \mathsf{z}_1)$ of honest P_1. Therefore, the output obtained by P_1 in Scenario 1 should also include z_0 (with all but negligible probability). Infact, we can argue that the output obtained by P_1 in Scenario 1 should in fact be $(\mathsf{z}_0, \mathsf{z}'_0)$ (with all but negligible probability) to be consistent with the ideal realization of f. This is because the simulator in Scenario 1 can induce an output comprising of z_0 only by invoking the ideal functionality with $x_3 = 0$ on behalf of corrupt P_3, which fixes the output of P_1 to include z'_0 as per the definition of f.

We can thus conclude that the output obtained by honest P_1 in Scenario 1 must be $(\mathsf{z}_0, \mathsf{z}'_0)$. Next, we consider another Scenario, say **Scenario 3** –

Scenario 3: Adversary corrupts P_1 but behaves honestly throughout the protocol. Suppose the input of honest P_3 is $x_3 = 1$.

First, it follows from the correctness of the protocol that since all parties including the corrupt parties behaved honestly in Scenario 3, the output computed must be in fact $(\mathsf{z}_1, \mathsf{z}'_1)$ computed on honest inputs, which is obtained by all (including the adversary). Next, we show an attack by the adversary controlling P_1 that allows her to obtain z'_0 as well, which violates security (as an adversary corrupting P_1 is not allowed to learn both inputs of honest P_2 i.e. z'_0 and z'_1, as per the ideal computation of f). The main idea is that the adversary simulates *in her head* Scenario 1, where there was a conflict between P_3 and P_2.

In the above execution of Scenario 3, let $m_{i\to j}$ denotes the peer-to-peer first-round message sent by P_i to P_j and b_i denotes the second-round broadcast message sent by P_i (it is without loss of generality to assume that the second-round messages are over broadcast; since private communication in the second round can be realized by exchanging public keys in the first round).

Round 1: On behalf of P_3, the adversary chooses input $x_3 = 0$ and some chosen randomness, say r_3. Using these values, the adversary recomputes the outgoing first-round peer-to-peer message from P_3 to P_1, say $\overline{m_{3\to 1}}$. However, the other first-round peer-to-peer messages i.e. $m_{3\to 2}, m_{2\to 3}, m_{2\to 1}, m_{1\to 2}$ and $m_{1\to 3}$ are fixed to be the same as what were received during the execution of Scenario 3.

Round 2: Next, the adversary recomputes the second-round broadcast message of P_3, say $\overline{b_3}$ as follows: Compute the broadcast message based on protocol specifications when P_3 did not receive any first-round peer-to-peer message from P_2. Note that this message can be computed using input $x_3 = 0$, randomness r_3 and the first-round peer-to-peer message $m_{1\to 3}$ received by P_3 from P_1 (which the adversary knows). The broadcast message of P_1, say $\overline{b_1}$ is recomputed based on honest input and randomness of P_1, the above simulated first-round peer-to-peer message $\overline{m_{3\to 1}}$ and $m_{2\to 1}$. Lastly, the broadcast message of P_2 is fixed to b_2 (same as received in the execution).

We observe that the above simulation in her head, allows the adversary to obtain a view that is identically distributed to the view of honest P_1 in Scenario 1. This is because both the simulation as well as Scenario 1 involve the messages $\overline{m_{3\to 1}}$, $\overline{b_1}$ and $\overline{b_3}$ being based on $x_3 = 0$. We thus infer that the adversary should be able to compute the output of Scenario 1 as well.

Since the output of Scenario 1 is $(\mathsf{z}_0, \mathsf{z}'_0)$, we can conclude that the adversary of Scenario 3 learns both z'_0 (via the simulation in her head) and z'_1 (via the output of the execution) which violates security, since this is not allowed as per the ideal computation of f.

Lastly, we note that the above proof can be extended to $n \leq 3t$ using player partitioning technique (An n-party protocol Π' tolerating $t \geq n/3$ corruptions can be transformed into a 3-party protocol Π tolerating 1 corruption, by making a party in Π emulate the protocol steps of t parties in Π').

Theorem 5 (BC-P2P, UA, CRS, $t > 1$). *There exist functions f such that no n-party two-round protocol can compute f with unanimous abort against $t > 1$ corruptions while making use of broadcast only in the first round (i.e. where the first round uses both broadcast and peer-to-peer channels[10] and second round uses only peer-to-peer channels).*

The proof appears in the full version [10]

References

1. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Two round information-theoretic MPC with malicious security. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11477, pp. 532–561. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_19
2. Badrinarayanan, S., Miao, P., Mukherjee, P., Ravi, D.: 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
3. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34961-4_10
4. Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 500–532. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_17
5. Chen, M., Cohen, R., Doerner, J., Kondi, Y., Lee, E., Rosefield, S., Shelat, A.: Multiparty generation of an RSA modulus. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12172, pp. 64–93. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_3
6. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: 18th Annual ACM Symposium on Theory of Computing, pp. 364–369. ACM Press, Berkeley, CA, USA, 28–30 May 1986. https://doi.org/10.1145/12130.12168
7. Cohen, R., Garay, J., Zikas, V.: Broadcast-optimal two-round MPC. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12106, pp. 828–858. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45724-2_28
8. Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 466–485. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_25
9. Damgård, I., Magri, B., Ravi, D., Siniscalchi, L., Yakoubov, S.: Broadcast-optimal two round MPC with an honest majority. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 155–184. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_6
10. Damgård, I., Ravi, D., Siniscalchi, L., Yakoubov, S.: Minimizing setup in broadcast-optimal two round MPC. Cryptology ePrint Archive, Report 2021/241 (2022). https://eprint.iacr.org/2022/293

11. Ganesh, C., Patra, A.: Broadcast extensions with optimal communication and round complexity. In: Giakkoupis, G. (ed.) 35th ACM Symposium Annual on Principles of Distributed Computing. pp. 371–380. Association for Computing Machinery, Chicago, IL, USA, 25–28 July 2016. https://doi.org/10.1145/2933057.2933082
12. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 468–499. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_16
13. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: On 2-round secure multiparty computation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 178–193. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_12
14. Goel, A., Jain, A., Prabhakaran, M., Raghunath, R.: On communication models and best-achievable security in two-round MPC. In: TCC, p. 690 (2021)
15. Dov Gordon, S., Liu, F.-H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_4
16. Dov Gordon, S., Liu, F.-H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_4
17. Ishai, Y., Kumaresan, R., Kushilevitz, E., Paskin-Cherniavsky, A.: Secure computation with minimal interaction, revisited. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 359–378. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_18
18. Ishai, Y., Kushilevitz, E., Paskin, A.: Secure multiparty computation with minimal interaction. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 577–594. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_31
19. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
20. Patra, A., Ravi, D.: On the exact round complexity of secure three-party computation. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 425–458. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_15