

# Broadcast-Optimal Two Round MPC with Asynchronous Peer-to-Peer Channels

Ivan Damgård<sup>1</sup>, Divya Ravi<sup>1 \*</sup>, Luisa Siniscalchi<sup>2</sup>, and Sophia Yakoubov<sup>1†</sup>

<sup>1</sup> Aarhus University, Denmark; {ivan, divya, sophia.yakoubov}@cs.au.dk

<sup>2</sup> Danish Technical University; luisi@dtu.dk

**Abstract.** In this paper we continue the study of two-round broadcast-optimal MPC, where broadcast is used in one of the two rounds, but not in both. We consider the realistic scenario where the round that does not use broadcast is *asynchronous*. Since a first asynchronous round (even when followed by a round of broadcast) does not admit any secure computation, we introduce a new notion of asynchrony which we call  $(t_d, t_m)$ -asynchrony. In this new notion of asynchrony, an adversary can delay or drop up to  $t_d$  of a given party's incoming messages; we refer to  $t_d$  as the *deafness threshold*. Similarly, the adversary can delay or drop up to  $t_m$  of a given party's outgoing messages; we refer to  $t_m$  as the *muteness threshold*.

We determine which notions of secure two-round computation are achievable when the first round is  $(t_d, t_m)$ -asynchronous, and the second round is over broadcast. Similarly, we determine which notions of secure two-round computation are achievable when the first round is over broadcast, and the second round is (fully) asynchronous. We consider the cases where a PKI is available, when only a CRS is available but private communication in the first round is possible, and the case when only a CRS is available and no private communication is possible before the parties have had a chance to exchange public keys.

**Keywords:** Secure Computation, Round Complexity, Asynchrony

---

\*Funded by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC).

†Funded by the Danish Independent Research Council under Grant-ID DFF-2064-00016B (YOSO).

# Table of Contents

BROADCAST-OPTIMAL TWO ROUND MPC WITH ASYNCHRONOUS PEER-TO-PEER CHANNELS .....	1
<i>Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov</i>	
1 Introduction .....	1
2 Secure Multiparty Computation (MPC) Definitions .....	9
3 P2P-BC .....	11
4 BC-P2P .....	40
A Preliminaries .....	48
B Non-Interactive One-or-Nothing Secret Sharing of Damgård <i>et al.</i> [DMR <sup>+</sup> 21] .....	55
C One-or-Nothing Secret Sharing with Intermediaries of Damgård <i>et al.</i> [DRSY23] .....	58
D Formal description of $\Pi_{\text{sl-abort}}$ .....	64

## 1 Introduction

Round complexity is an important metric of the efficiency of a secure computation (MPC) protocol. When MPC is run over a high latency network, each round of communication can take a long time. Two rounds has been shown to be optimal; one round of communication is clearly not enough for secure computation, since it leaves the protocol vulnerable to *residual function attacks*, where the adversary can recompute the function with the same honest party inputs and different corrupt party inputs simply by preparing different messages on the corrupt parties' behalf.

However, optimal round complexity is only the first step towards efficient use of resources. *Broadcast*, which, in practice, requires either multiple rounds of peer-to-peer communication or special channels (which use, e.g., physical assumptions or blockchains), is itself an expensive resource. Most known two-round MPC protocols either use broadcast in both rounds [GLS15,GS18,BL18], or only achieve the weakest security guarantee (selective abort) [ACGJ19]. Cohen *et al.* [CGZ20], Damgård *et al.* [DMR<sup>+</sup>21] and Damgård *et al.* [DRSY23] explore the tradeoffs between the security of two-round MPC protocols and their use of broadcast (where broadcast can be used in the first round only, in the second round only, in both rounds, or in neither round). Cohen *et al.* focus on the dishonest majority setting; Damgård *et al.* [DMR<sup>+</sup>21] focus on the honest majority setting; and Damgård *et al.* [DRSY23] additionally remove the use of PKI and private peer-to-peer channels in the first round from the previous works. All three of these papers gave tight characterizations of the security guarantees that can be achieved in the different settings.

## 1.1 Our Contributions

In this paper, we focus on the realistic setting where the rounds that do not use broadcast are also not fully synchronous. Since fully asynchronous MPC has been studied in the literature extensively [BCG93,BKR94,HNP05,HNP08], we limit ourselves to the case where at least one of the two rounds uses a synchronous broadcast channel.

*Asynchrony in the First Round* If the first round is fully asynchronous, the adversary can prevent a party from communicating anything to any of the honest parties in the first round; since correctness demands that the protocol produce an output even in the event of adversarial message scheduling, this means that either that party’s input cannot influence the output (this is known as *input deprivation*), or that the protocol is vulnerable to residual function attacks (since the adversary can, like in one-round protocols, recompute the second-round messages on behalf of a corrupt party who no-one heard from in the first round).

So, we introduce a new flavor of asynchrony, where the adversary is only able to delay *up to a certain threshold* of messages to and from any one party. We call it  $(t_d, t_m)$ -asynchrony. In  $(t_d, t_m)$ -asynchrony, at most  $t_m$  of any party’s messages can be arbitrarily delayed or dropped, where  $t_m$  is the *muteness threshold*. Similarly, at most  $t_d$  of the messages to a given party can be arbitrarily delayed or dropped, where  $t_d$  is the *deafness threshold*. We allow the adversary to be *rushing* i.e. determine which messages to delay or drop based on the messages she sees during the round. By setting  $t_m$ , we ensure that each party communicates to sufficiently many parties in the first round, enabling us to sidestep the problem of input deprivation.

This new notion of asynchrony is a contribution in and of itself. It is incomparable to the standard notion of asynchrony, where the adversary can arbitrarily delay — *but not drop* — any number of the messages<sup>3</sup>.

We now summarize our findings about two-round MPC where the first round is  $(t_d, t_m)$ -asynchronous (over peer-to-peer channels), and the second round uses synchronous broadcast. Let  $n$  be the number of participants, and  $t$  be the corruption threshold. We show that, if a PKI is available, no such secure two-round MPC is possible if  $n \leq t + t_m$ . When  $t + t_m < n \leq 2t + t_m$ , the best achievable guarantee is *unanimous abort*, where honest parties either all learn the output, or abort. When  $2t + t_m < n$ , identifiable abort — where in the event of an abort, honest parties agree on the identity of a corrupt party — is additionally achievable. (Stronger guarantees have already been ruled out if broadcast is not available in the first round, as long as  $t > 1$  [DMR<sup>+</sup>21].) Our constructions that rely on a PKI use one-or-nothing secret sharing [DMR<sup>+</sup>21] (which is a flavor of

---

<sup>3</sup>Our notion is also incomparable to the notion of send / receive-omission corruptions of [ZHM09] which considers an adversary who can send-corrupt *some* parties whose (any number of) sent messages may be dropped and / or receive-corrupt *some* parties that may not receive (any of the) messages sent to them. This is different from our notion where a *bounded* number of outgoing and incoming messages for *each* party is blocked.

secret sharing that allows a dealer to share a vector of secrets, among which at most one secret would be reconstructed).

If a PKI is not available, but parties are able to send one another private messages in the first round, as before no such secure two-round MPC is possible if  $n \leq t + t_m$ . Additionally, nothing is achievable if  $n \leq t + 2t_d$  and  $n \leq t + 2t_m$ . However, the rest of the time, unanimous abort is possible. Identifiable abort is unachievable if  $n \leq 3t + t_m$ ; we show that it is achievable if  $3t + t_m < n$  and  $\min(t_d, t_m) \leq t$ , but we leave what happens without that last requirement as an open problem.

If neither a PKI nor private channels are available in the first round, we show that no such secure two-round MPC is possible if  $n \leq t + t_d + t_m$ . However, the rest of the time, unanimous abort is achievable. As before, identifiable abort is unachievable if  $n \leq 3t + t_m$ ; we show that it is achievable if  $3t + t_m < n$  and  $3t + t_d < n$ , but we leave what happens without that last requirement as an open problem.

We give several constructions that do not rely on a PKI. For somewhat looser bounds, we show constructions that rely on standard assumptions by generalizing the one-or-nothing secret sharing with intermediaries introduced by Damgård *et al.* [DRSY23]. We provide also new constructions with the tightest bounds of  $t, t_d$  and  $t_m$  that rely on differing-inputs obfuscation to demonstrate feasibility, or, rather, the infeasibility of a negative result. (The obfuscation-based constructions that achieve identifiable abort are additionally limited to a constant number of parties.)

*Asynchrony in the Second Round* If the first round uses fully synchronous broadcast, security is possible even if the second round is asynchronous in the classical sense; that is, the adversary can arbitrarily delay (but not drop) any number of the second-round messages. In this setting, we show that if a PKI is available, no secure MPC is possible if  $n \leq 2t$ . However, the strongest guarantee — guaranteed output delivery — is achievable otherwise, as shown by a simple observation by Rambaud and Urban [RU21].

If a PKI is not available, but parties are able to send one another private messages in the first round, selective abort is achievable as long as  $2t < n$ . No stronger guarantee is achievable, by the lower bounds of Patra and Ravi [PR18] and Damgård *et al.* [DRSY23].

If neither a PKI nor private channels are available in the first round, we show that no secure MPC is possible for any corruption threshold  $t \geq 1$ .

## 1.2 Terminology

We characterize our protocols in terms of (a) the kinds of communication channels used 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 consider different kinds of channels:

**Broadcast Channels (BC)**, where each broadcast message recipient has the guarantee that all other recipients received the same message.

**Peer to Peer Channels (P2P)**, where recipients have no guarantee of consistency.

When a PKI is available, or when the parties have already had a chance to exchange encryption keys, *private* communication is possible over both BC and P2P channels. However, when this is not the case (that is, when a PKI is not available, and this is the first round), it makes sense to break these up into the following:

**Public Peer to Peer Channels (PubP2P)**, where recipients don't have the guarantee that all others see the same message (nor do they have a guarantee of privacy).

**Private Peer to Peer Channels (PrivP2P)**, where recipients don't have the guarantee that all others see the same message, but parties can communicate messages privately.

**Public Broadcast Channels (PubBC)**, where a party can either broadcast a message *or* communicate it over public peer to peer channels. (Note that using a broadcast channel is strictly stronger than using a public peer to peer channel; the only reason to choose to use a public peer to peer channel instead of a broadcast channel is efficiency.)

**Broadcast with Private Channels (PrivBC)**, where a party can either broadcast a message *or* communicate it privately.

We use a concatenation of two channel names to denote the communication structure of a protocol. As an example, PrivP2P-BC denotes a protocol whose first round is over private peer to peer channels, and whose second round is over broadcast. (Private messages are also possible in the second round, since the parties can exchange public keys in the first round.)

*Security Guarantees* An MPC protocol can achieve one of five notions of security. These are described below, from weakest to strongest (with the exception that fairness and identifiable abort are incomparable).

**Selective Abort (SA)**: Every honest party either obtains the output, or aborts.

**Unanimous Abort (UA)**: Either *all* honest parties obtain the output, or they all (unanimously) abort.

**Identifiable Abort (IA)**: Either all honest parties obtain the output, or they all (unanimously) abort identifying one corrupt party.

**Fairness (FAIR)**: Either all parties obtain the output, or none of them do. (An adversary should not be able to learn the output if the honest parties do not.)

**Guaranteed Output Delivery (GOD)**: All honest parties will learn the computation output no matter what the adversary does.

*Setup* We consider two kinds of setup: either only a common reference string (CRS), where parties have access to a common string generated in a trusted way, or both a CRS and a (trusted) PKI, where parties additionally know one another’s public keys before the protocol starts.

### 1.3 Technical Overview

We consider protocols with and without a PKI. With a PKI, we consider the P2P-BC and BC-P2P settings; without a PKI, we consider the PrivP2P-BC, PubP2P-BC, PrivBC-P2P and PubBC-P2P settings. We explore what security guarantees are achievable when the P2P and PrivP2P rounds are asynchronous.

**Asynchrony in the First Round** As we explained earlier, in the P2P-BC and PrivP2P-BC settings, no security guarantee is achievable when the adversary can schedule the first-round messages arbitrarily. However, if we make some restrictions on the message scheduling, some notions of security become achievable for some thresholds. To this end, we introduce  $(t_d, t_m)$ -asynchrony, where the adversary can drop or delay only  $t_d$  incoming messages for each party, and  $t_m$  outgoing messages for each party.

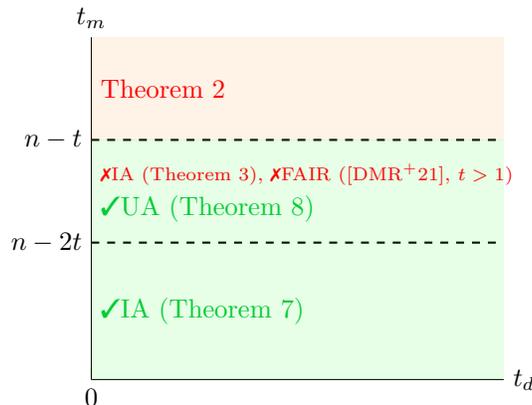


Fig. 1: Partial Asynchrony Feasibility and Impossibility Results in the P2P-BC setting, with a PKI and CRS

Prior works [Cle86,DMR<sup>+</sup>21] show that even in the synchronous setting (with  $t_d = t_m = 0$ ) and given a PKI, no P2P-BC or PrivP2P-BC protocol can achieve fairness or GOD<sup>4</sup>. Figures 1 and 2 describe our findings about the feasibility of

<sup>4</sup>The impossibility holds for more general settings such as when  $t > 1$  or  $n \leq 3t$ . However, it is possible to achieve GOD for the special case when  $t = 1$  and  $n \geq 4$  [IKP10], [IKKP15] (even in the P2P-P2P synchronous setting with no CRS or PKI).

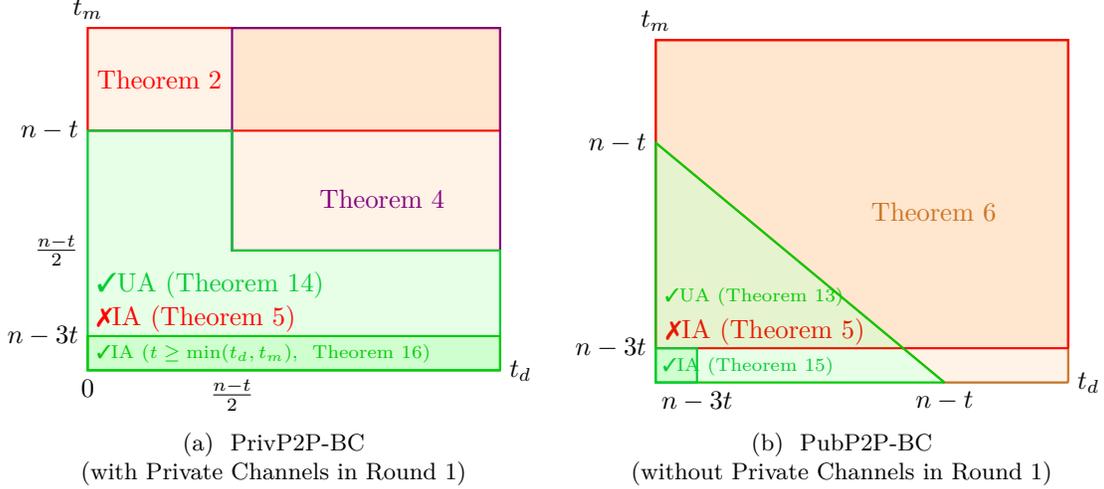


Fig. 2: Partial Asynchrony Feasibility and Impossibility Results with a CRS.

SA, UA and IA when a PKI is available, when a PKI is not available but private channels in the first round are, and when neither a PKI nor private channels in the first round are available.

*Lower Bounds* Most of our negative results follow a common blueprint. We start by showing (in Theorem 1) that if there is a group of parties  $A$  whose second-round messages do not depend on messages from a disjoint group  $B$ , and if the adversary has all the information she needs in order to recompute the first and second round messages of  $B$  and other messages that *depend* on messages from  $B$  (while keeping the messages — and thus inputs — of  $A$  fixed), then the adversary is able to execute a residual function attack by recomputing the function with different inputs on behalf of  $B$ . We then show how adversarial network scheduling and corruption strategies can lead to such groups in various settings.

In the P2P-BC, SA, PKI,  $t_m \geq n-t, t_d \geq 1$  setting, the messages of one corrupt party  $B = \{P_i\}$  to the set  $A$  of honest parties (of whom there are  $n-t \leq t_m$ ) need not be delivered. The adversary can then fix the input of the honest parties and the rest of the corrupt parties, and recompute the first round messages of  $P_i$  (and the subsequent second round messages of all corrupt parties) based on various inputs, resulting in a residual function attack. This shows that in this setting, even SA cannot be achieved (Theorem 2).

In the P2P-BC, IA, PKI,  $t_m \geq n-2t, t_d \geq 1$  setting, if one corrupt party  $B = \{P_i\}$  does not send any messages to a set  $A$  of  $t$  honest parties,  $P_i$  cannot

We leave open the question of weakening the synchrony assumptions for these special cases.

be identified as a cheater by the other parties, since they do not know whether  $P_i$  is truly to blame, or whether  $A$  is corrupt and are lying about not having heard from  $P_i$ . So, output must still be computed. The messages from  $P_i$  to the remaining  $n - 2t \leq t_m$  honest parties can be dropped, so now, messages from  $P_i$  are only delivered to other corrupt parties. As before, the adversary can then fix the input of the honest parties and the rest of the corrupt parties, and recompute the first round messages of  $P_i$  (and the subsequent second round messages of all corrupt parties) based on various inputs. This shows that in this setting, IA cannot be achieved (Theorem 3).

In the PrivP2P-BC, SA, CRS,  $n \leq t + 2t_d$ ,  $n \leq t + 2t_m$  setting, the adversary can partition the honest parties into sets  $A$  and  $B$  of equal size that do not communicate at all in the first round, since  $|A| = |B| = \frac{n-t}{2} \leq t_d$  and  $|A| = |B| = \frac{n-t}{2} \leq t_m$ . Now, since a PKI is not available (and thus the parties in  $B$  have no long-term secrets), the adversary can simulate  $B$  with arbitrary inputs, and recompute the corresponding messages of the corrupt parties. Since the messages of parties in  $A$  need not change, this is an effective residual function attack. This shows that in this setting, even SA cannot be achieved (Theorem 4).

In the PubP2P-BC, SA, CRS,  $n \leq t + t_d + t_m$  setting, the adversary can partition the honest parties into sets  $A$  and  $B$  where  $|A| = t_m$  and  $|B| = t_d$ , and where  $A$  does not hear from  $B$  in the first round. Now, since a PKI is not available (and thus the parties in  $B$  have no long-term secrets), the adversary can simulate  $B$  with arbitrary inputs (using the public messages from  $A$  to  $B$  as part of the view of  $B$ , which affects  $B$ 's second round messages), and recompute the corresponding messages of the corrupt parties. This shows that in this setting, even SA cannot be achieved (Theorem 6).

The PrivP2P-BC, IA, CRS,  $n \leq 3t + t_m$  setting is the only one for which we use a different blueprint. This proof follows the proof of Damgård *et al.* [DRSY23], which shows that in the fully synchronous PrivP2P-BC, IA, CRS,  $n \leq 3t$ , IA cannot be achieved. They do this by showing how  $t$  corrupt parties can get away with sending first round messages computed on different inputs to two disjoint sets of honest parties; they show that the protocol must yield output on *both* inputs, since the honest parties may not be able to identify a cheater. We show that the presence of an additional  $t_m$  parties, all of who may not have heard a given first round message due to network scheduling, does not help matters. This shows that in this setting, IA cannot be achieved (Theorem 5).

*Upper Bounds* Our upper bounds in the presence of a PKI follow the blueprint of the constructions of Cohen *et al.* [CGZ20], Damgård *et al.* [DMR<sup>+</sup>21] and Damgård *et al.* [DRSY23]. Damgård *et al.* introduce one-or-nothing secret sharing; for our constructions with PKI, we simply tweak the reconstruction thresholds of one-or-nothing secret sharing to account for the adversary's ability to drop  $t_m$  of each party's outgoing messages.

When a PKI is *not* available, we show protocols using variant of one-or-nothing secret sharing defined by Damgård *et al.* [DRSY23], which is called *one-or-nothing secret sharing with intermediaries*. Here we adjust the privacy as well as reconstruction thresholds to adapt them to the  $(t_d, t_m)$ -asynchronous setting.

Unfortunately, these constructions without PKI are not tight with respect to our lower bounds. This led us to bring our study forward and investigate if it was possible to obtain tighter lower bounds or alternatively, design matching upper bounds that are based on stronger assumptions (as this would give evidence that proving a tighter lower bound is not possible).

Towards this, we provide completely new constructions based on differing-inputs obfuscation (diO) [BGI<sup>+</sup>01,ABG<sup>+</sup>13]. These constructions rely on a CRS in the form of an obfuscated program the code of which hides a secret decryption key. In the first round, each party encrypts her input to the corresponding public encryption key, generates a signing - verification key pair, signs her ciphertext, and sends the ciphertext, verification key and signature to all of her peers. In the second round, the parties echo everything they heard in the first round over the broadcast channel; these echos then serve as input to the obfuscated program. The program checks that the echos are consistent enough before decrypting the ciphertexts and evaluating the function. If the echos are *not* consistent enough, it aborts; to achieve identifiable abort it is possible to use the conflict graph that it derives from the echos to identify a cheater. One difficulty is that we must make sure that the adversary cannot copy and reuse an honest party’s ciphertext (with a fresh verification key). We do this by introducing a new primitive which we call *puncturable public key encryption*. Here, the sender uses her own public key as an input to the encryption algorithm, so that the resulting ciphertext is bound to the sender. The receiver’s public key can be *punctured* at one or more senders’ public keys, so that ciphertexts produced by those senders no longer carry any information about the messages used.

Of course, diO is wildly impractical, and as mentioned previously, our results which use diO should be seen as feasibility results (or rather, as evidence of the infeasibility of proving a tighter lower bound).

**Asynchrony in the Second Round** For BC-P2P protocols, the classical notion of asynchrony in the second round does not preclude secure computation, so we stick to that. We summarize our findings about asynchronous BC-P2P protocols in Table 1.

In this setting, we show that for  $n \leq 2t$ , a BC-P2P protocol cannot even achieve selective abort, even if a PKI is available (Theorem 17)<sup>5</sup>. This follows from the fact that parties can’t wait for more than  $n - t \leq t$  second-round messages before computing output, since the remaining  $\leq t$  parties may be corrupt and may not have sent messages. So, parties must be able to compute output even if they only received second-round messages from corrupt parties; this allows the adversary to recompute the output based on different corrupt parties’ inputs, in what is an effective residual function attack.

For  $2t < n$ , given a PKI, a BC-P2P protocol can achieve the strongest notion of security — GOD. This follows from an observation in [RU21] (that we state in

---

<sup>5</sup>It already followed from the work of Cohen *et al.* that unanimous abort is unachievable in this setting.

$t$	Setup & Communication Pattern	selective abort	unanimous abort	identifiable abort	fairness	guaranteed output delivery
$n \leq 2t$	CRS + PKI, BC-P2P	$\times$ (Theorem 17)	$\times$ [CGZ20]	$\rightarrow \times$	$\times$ [Cle86]	$\rightarrow \times$
$2t < n$	CRS + PKI, BC-P2P	$\checkmark \leftarrow$	$\checkmark \leftarrow$	$\checkmark \leftarrow$	$\checkmark \leftarrow$	$\checkmark$ Obs 4 [RU21]
$2t < n$	CRS, PrivBC-P2P	$\checkmark$ (Theorem 20)	$\times$ [PR18], [DRSY23]	$\rightarrow \times$	$\rightarrow \times$	$\rightarrow \times$
$2t < n$	CRS, PubBC-P2P	$\times$ (Theorem 19)	$\rightarrow \times$	$\rightarrow \times$	$\rightarrow \times$	$\rightarrow \times$

Table 1: Feasibility and impossibility of partially-asynchronous BC-P2P MPC with different guarantees. The first-round broadcast and peer-to-peer communication is synchronous and second communication is asynchronous. Existing impossibility results in the BC-P2P setting where the second round is synchronous extend to the BC-P2P setting where the second round is asynchronous. 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.

Obs 4) that shows that the BC-P2P construction of Damgård *et al.* [DMR<sup>+</sup>21] works even if the second round is asynchronous.

Without a PKI, a PrivBC-P2P protocol can achieve selective abort as long as  $2t < n$ , which is the best guarantee we can hope for (Theorem 20)<sup>4</sup>. (Nothing is possible in the PubBC-P2P setting (Theorem 19)). For our SA construction, we rely on certain properties of synchronous schemes to extend them to this setting. We show that any *synchronous* protocol that is PrivBC-P2P SA CRS  $2t < n$  could be also executed with an asynchronous second round as long as it is easy to determine whether second-round messages are “valid”, the inputs can be extracted from first-round messages, and  $n - t$  valid second-round messages are sufficient to recover the output. We then adapt the construction of Ananth *et al.* [ACGJ18], using commitments and NIZKs, to provide an instantiation of the starting PrivBC-P2P synchronous protocol.

## 1.4 Organization

In Section 2, we define our MPC security notion and the notation we need for our setting. In Section 3, we describe our lower and upper bounds for the setting when the first round is over  $(t_d, t_m)$ -asynchronous channels. In Section 4, we describe our lower and upper bounds for the setting when the second round is asynchronous.

## 2 Secure Multiparty Computation (MPC) Definitions

In this section we recall the relevant 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.<sup>6</sup>

*Real-world.* An  $n$ -party protocol  $\Pi = (P_1, \dots, P_n)$  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.

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. 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  $\Pi = (P_1, \dots, P_n)$  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  $\Pi$  under  $(\mathcal{A}, \mathcal{I})$  in the real world, on input vector  $x = (x_1, \dots, x_n)$ , auxiliary input  $\mathbf{aux}$  and security parameter  $\lambda$ , denoted  $\text{REAL}_{\Pi, \mathcal{I}, \mathcal{A}(\mathbf{aux})}(x, \lambda)$ , is defined as the output vector of  $P_1, \dots, P_n$  and  $\mathcal{A}(\mathbf{aux})$  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  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{id-abort}, \text{fairness}, \text{god}\}$ . Let  $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$  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  $x = (x_1, \dots, x_n)$ , auxiliary input  $\mathbf{aux}$  to  $\mathcal{S}$  and security parameter  $\lambda$ , denoted  $\text{IDEAL}_{f, \mathcal{I}, \mathcal{S}(\mathbf{aux})}^{\text{type}}(x, \lambda)$ , is defined as the output vector of  $P_1, \dots, P_n$  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  $(y_1, \dots, y_n) = f(x'_1, \dots, x'_n)$ . If there are no corrupt parties or  $\text{type} = \text{god}$ , proceed to step 4.*

---

<sup>6</sup>We note that our security proofs can translate to an appropriate (synchronous) composable setting with minimal changes.

- (a) If  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{id-abort}\}$ : The trusted party sends  $\{y_i\}_{i \in \mathcal{I}}$  to  $\mathcal{S}$ .
- (b) If  $\text{type} = \text{fairness}$ : The trusted party sends `ready` to  $\mathcal{S}$ .
- 3. Simulator  $\mathcal{S}$  responds to trusted party:
  - (a) If  $\text{type} = \text{sl-abort}$ : 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  $(\text{abort}, \mathcal{J})$  to the trusted party.
  - (b) If  $\text{type} \in \{\text{un-abort}, \text{fairness}\}$ : The simulator can send `abort` to the trusted party. If it does, we take  $\mathcal{J} = [n] \setminus \mathcal{I}$ .
  - (c) If  $\text{type} = \text{id-abort}$ : If it chooses to abort, the simulator  $\mathcal{S}$  can select a corrupt party  $i^* \in \mathcal{I}$  who will be blamed, and send  $(\text{abort}, i^*)$  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  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{fairness}\}$ , we let `abortmsg` =  $\perp$ .
      - if  $\text{type} = \text{id-abort}$ , 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  $\text{type} = \text{god}$ , 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  $\text{type} \in \{\text{sl-abort}, \text{un-abort}, \text{id-abort}, \text{fairness}, \text{god}\}$ . Let  $f : (\{0, 1\}^*)^n \rightarrow (\{0, 1\}^*)^n$  be an  $n$ -party function. A protocol  $\Pi$   $t$ -securely computes the function  $f$  with  $\text{type}$  security if for every PPT real-world adversary  $\mathcal{A}$  with auxiliary input  $\text{aux}$ , 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

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

### 3 P2P-BC

In this section, we assume that the first round of communication occurs over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony (and the peer-to-peer and broadcast communication in the second round is fully synchronous). We determine the feasibility of various notions of security, in three settings: (1) when a PKI is available, (2) when no PKI is available, but a CRS and private peer-to-peer channels are, and (3) when no PKI or private channels are available in the first round, but a CRS is. Our results are summarized in Figures 1 and 2.

### 3.1 Lower Bounds

Before describing our lower bounds, we present a theorem that is useful for our lower bound arguments. In this theorem, we identify what type of protocol design makes it vulnerable to a residual attack by the adversary i.e. allows the adversary to obtain the output on multiple inputs of her choice, while keeping the inputs of a subset of honest parties fixed.

**Theorem 1.** *Consider an  $n$ -party two-round protocol  $\Pi$ . Let  $E$  be the event that the parties can be assigned to three sets  $A, B$  and  $C$  with the following properties:*

1.  *$A$  contains a subset of honest parties.*
2.  *$C$  contains the set of corrupt parties, where  $1 \leq |C| \leq t$ .*
3.  *$|A|, |B| \geq 1$ , and  $A$  is disjoint from  $B \cup C$  (but  $B$  and  $C$  are not necessarily disjoint).*
4. *The second-round messages of parties in  $A$  do not depend on the first-round messages of parties in  $B$ .*
5. *The adversary (controlling the parties in  $C$ ) has access to the communication from  $A$  to  $B$  as well as private information that parties in  $B$  receive from the setup (if any).*
6. *The adversary obtains the output (computed on honest parties' inputs).*

*If  $\Pi$  allows the above event  $E$  to occur with non-negligible probability, then there exists functions  $f$  such that  $\Pi$  can not securely compute  $f$ .*

*Proof.* We use the function  $f_{\text{mot}}$ . Let the input of  $P_n$  be a pair of strings  $x_n = (z_0, z_1)$ , where  $z_0, z_1 \in \{0, 1\}^\lambda$ , and let the input of every other party  $P_i$  ( $i \in \{1, \dots, n-1\}$ ) be a single bit  $x_i \in \{0, 1\}$ .  $f_{\text{mot}}$  allows everyone to learn  $z_c$  where  $c = \bigoplus_{i=1}^{n-1} x_i$ .

Towards a contradiction, we assume that the two-round secure protocol  $\Pi$  computes  $f_{\text{mot}}$  securely. Suppose that event  $E$  (described in the theorem) occurs during an execution of  $\Pi$  and results in the adversary obtaining the output (computed on honest parties' inputs).

We observe that, since the second-round messages of parties in  $A$  do not depend on the first round messages of parties in  $B$ , these are independent of the inputs of parties in  $B$ . This makes  $\Pi$  susceptible to the following residual attack: the adversary can use different choices of inputs on behalf of the parties in  $B$  and recompute their first and second round messages, while keeping the messages of parties in  $A$  fixed. The adversary also keeps the first-round messages of parties in  $C \setminus B$  fixed and recomputes their second-round messages (based on the recomputed first-round messages of  $B$ ).

Note that recomputing messages on behalf of parties in  $B$  (based on chosen inputs) requires (a) the private information (if any) that parties in  $B$  receive from the setup, and (b) the first-round messages that parties in  $B$  received from parties in  $A$  and  $C$ . Since the adversary has access to all of this (based on the assumptions and the fact that the adversary controls  $C$ ), it is possible for the adversary to recompute messages on behalf of parties in  $B$  (using chosen inputs). It is now easy to see that this will allow the adversary to obtain multiple

evaluations of the function, for different choices of inputs of parties in  $B$  while the inputs of other parties remain fixed. This contradicts the security of  $\Pi$ . More concretely, suppose  $P_n \in A$ . Then, this “residual attack” breaches the privacy property of the protocol, as it allows the adversary to learn both input strings of an honest  $P_n$  (which is not allowed in an ideal realization of  $f_{\text{mot}}$ ).

**Corollary 1.** *Assume a setup with PKI or correlated randomness and the existence of a protocol  $\Pi$  where properties (1), (2), (3), (4) and (6) of  $E$  described in Theorem 1 are satisfied with non-negligible probability. If  $B \subseteq C$  holds, then there exist functions  $f$  such that  $\Pi$  cannot securely compute  $f$ .*

*Proof.* We observe that when  $B \subseteq C$ , property (5) of Theorem 1 automatically holds – as the adversary controlling the parties in  $B$  has access to their private information received as a part of the setup and incoming messages. Hence, the corollary holds.

**Corollary 2.** *Assume a setup with CRS, a network with private peer-to-peer channels and the existence of a protocol  $\Pi$  where properties (1), (2), (3), (4) and (6) of  $E$  described in Theorem 1 are satisfied with non-negligible probability. If there is no communication from parties in  $A$  to parties in  $B$ , then there exist functions  $f$  such that  $\Pi$  cannot securely compute  $f$ .*

*Proof.* It is easy to see that if there is no communication from parties in  $A$  to  $B$  and the setup is public, then property (5) of Theorem 1 is satisfied by default. Hence, the corollary holds.

**Corollary 3.** *Assume a setup with CRS, a network with public peer-to-peer channels and the existence of a protocol  $\Pi$  where properties (1), (2), (3), (4) and (6) of  $E$  described in Theorem 1 are satisfied with non-negligible probability. Then, there exist functions  $f$  such that  $\Pi$  cannot securely compute  $f$ .*

*Proof.* Property (5) of Theorem 1 must also hold since the setup is public and the public peer-to-peer channels enable the adversary to learn the incoming messages of parties in  $B$ . Hence, the corollary holds.

**With PKI** In this section, we assume the availability of a PKI. We adopt the same proof approach in each of our negative results in this section: we describe an adversarial strategy and message scheduling that results in the occurrence of the event  $E$  described in Cor 1. We then invoke the impossibility result of Cor 1 to complete the proof.

**Theorem 2 (P2P-BC, SA, PKI,  $t_m \geq n - t$ ,  $t_d \geq 1$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with selective abort if  $t_m \geq n - t$  and  $t_d \geq 1$ ; where the first round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is over synchronous broadcast and peer-to-peer channels.*

*Proof.* We use the function  $f_{\text{mot}}$  described in Theorem 1. Towards a contradiction, we assume a protocol  $\Pi$  computing  $f_{\text{mot}}$  with selective abort exists, whose first round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and second round communication is over synchronous broadcast and peer-to-peer channels.

Consider a scenario where the adversary passively corrupts the parties in a set  $C$  (where  $|C| = t$ ) and  $A$  denotes the set of honest parties. Since the first round is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, the adversary can schedule the first round messages of a party in  $C$ , say  $P_i$ , such that they are received by only corrupt parties. Such a scheduling is allowed, since the messages of  $P_i$  are delivered to parties in  $C$  where  $|C| = t \geq n - t_m$  parties, and each honest party hears from  $n - 1 \geq n - t_d$  parties. The correctness of  $\Pi$  (which must hold as everyone including the passively controlled parties behaved honestly) dictates that this execution must result in output computation. However, since none of the parties in  $A$  received the first round messages of  $P_i$ , it is easy to see that their second-round messages are independent of  $P_i$ 's first round message. Setting  $B = \{P_i\} \subseteq C$ , it is easy to check that each of the conditions of Cor 1 hold. It now directly follows from Cor 1 that  $\Pi$  cannot securely compute  $f_{\text{mot}}$ , completing the proof.

**Theorem 3 (P2P-BC, IA, PKI,  $t_m \geq n - 2t, t_d \geq 1$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with identifiable abort if  $t_m \geq n - 2t$  and  $t_d \geq 1$ ; where the first round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is over synchronous broadcast and peer-to-peer channels.*

*Proof.* We use the function  $f_{\text{mot}}$  described in Theorem 1. Towards a contradiction, we assume a protocol  $\Pi$  computing  $f_{\text{mot}}$  with identifiable abort exists, whose first round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and second round communication is over synchronous broadcast and peer-to-peer channels.

Consider a partition of the set of parties into three disjoint sets  $S_0, S_1$  and  $S_2$ , where  $|S_0| = t_m$ ,  $|S_1| = t$  and  $|S_2| \leq t$ . The adversary schedules the first round messages of a party in  $S_1$ , say  $P_i$ , such that they are not received by parties in set  $S_0$ . Such a scheduling is allowed, since the messages of  $P_i$  are delivered to everyone except the  $t_m$  parties in  $S_0$  and each party hears from at least  $n - 1 \geq n - t_d$  parties.

We now consider two scenarios:

**Scenario 1:** Adversary controls the party  $P_i$  in  $S_1$  who behaves as per protocol specifications except the following:

- In the first round,  $P_i$  does not send first-round messages to honest parties in  $S_2$ .
- In the second round,  $P_i$  pretends as if she did not receive first-round messages from parties in  $S_2$ . In other words,  $P_i$  sends second-round messages based on protocol specifications when  $P_i$  did not receive first-round messages from parties in  $S_2$ .

**Scenario 2:** Adversary controls the parties in  $S_2$  (where  $|S_2| \leq t$ ) who behave as follows:

- In the first round, parties in  $S_2$  behave honestly, except that they do not send their first-round message to  $P_i$ .
- In the second round, parties in  $S_2$  pretend as if they did not receive the first-round message from  $P_i$ .

The honest parties in  $S_0$  cannot distinguish between the above two scenarios, since in both scenarios  $P_i$  and parties in  $S_2$  claim that they have not received first-round messages from each other. Since the honest parties in  $S_0$  do not know whom to blame ( $P_i$  or the parties in  $S_2$ , where  $|S_2| \leq t$ ), it must be the case that both the above scenarios result in output computation (i.e. does not result in an abort).

Consider an execution of  $\Pi$  where the adversary controls the parties in  $S_1$  who behave honestly except a single party  $P_i \in S_1$  who behaves as per Scenario 1. Since none of the honest parties in  $S_0 \cup S_2$  received the first round messages of  $P_i$ , it is easy to see that their messages are independent of  $P_i$ 's input. Based on the above argument, this scenario must result in output computation. This output must also be learnt by the adversary since its view subsumes the view of an honest  $P_i$  in Scenario 2 (who learns the output).

Setting  $A = S_0 \cup S_2$  as the set of honest parties,  $B = \{P_i\}$  and  $C = S_1$  (where  $B \subseteq C$ ), one can check that each of the conditions of Cor 1 holds. It now directly follows from Cor 1 that  $\Pi$  cannot securely compute  $f_{\text{mot}}$ , completing the proof.

**Without PKI, With Private Channels** In this section, we present two negative results for the setting when PKI is not available but the peer-to-peer channels are private. For the first negative result, just like in the previous section, we adopt the approach of describing an adversarial strategy and message scheduling that results in the occurrence of the event  $E$  described in Cor 2. We then invoke the impossibility result of Cor 2 to complete the proof. The second negative result shows impossibility of identifiable abort when  $n \leq 3t + t_m$ . This proof is a slight modification of the proof of Damgård *et al.* [DRSY23], which shows the impossibility of identifiable abort in the synchronous P2P-BC setting when  $n \leq 3t$ .

**Theorem 4 (PrivP2P-BC, SA, CRS,  $n \leq t + 2t_d$ ,  $n \leq t + 2t_m$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with selective abort if  $n \leq t + 2t_d$  and  $n \leq t + 2t_m$ ; where the first round communication is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is over synchronous broadcast and peer-to-peer channels.*

*Proof.* We use the same function  $f_{\text{mot}}$  used in the proof of Theorem 1. Towards a contradiction, we assume a PrivP2P-BC, SA, CRS protocol  $\Pi$  with  $n \leq t + 2t_d$  and  $n \leq t + 2t_m$  securely computing  $f_{\text{mot}}$  exists, whose first-round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and second-round communication is over synchronous broadcast and peer-to-peer channels.

Consider a partition of the parties into three disjoint sets  $A, B$  and  $C$ , where  $|A| = \frac{n-t}{2}$ ,  $|B| = \frac{n-t}{2}$ , and  $|C| = t$ , respectively. Consider an execution of  $\Pi$  where the adversary *passively* corrupts the  $t$  parties in  $C$  and schedules the messages in the first round so that no messages from  $B$  are delivered to  $A$ , and vice versa. Such a scheduling is permitted based on the assumption that  $t_m \geq \frac{n-t}{2}$  and  $t_d \geq \frac{n-t}{2}$ .

The correctness of  $\Pi$  (which must hold as everyone including the passively controlled parties behaved honestly) dictates that this execution must result in output computation. However, we note that second-round messages of parties in  $A$  do not depend on the first-round messages of parties in  $B$  (as parties in  $A$  did not receive any first-round messages from parties in  $B$ ). Furthermore, since there is no first-round communication from parties in  $A$  to parties in  $B$  as well, we observe that all the conditions of Cor 2 are satisfied. It now directly follows from Cor 2 that  $\Pi$  cannot securely compute  $f_{\text{tot}}$ , completing the proof.

**Theorem 5 (PrivP2P-BC, IA, CRS,  $n \leq 3t + t_m, t_d \geq 1$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with identifiable abort if  $n \leq 3t + t_m$ ; where the first round communication is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is over synchronous broadcast and peer-to-peer channels.*

*Proof.* We use the function  $f_{\text{ot}}$  (also used in Theorem 1 of [DRSY23]), described below. Let the input of  $P_1$  and  $P_2$  be a pair of strings  $x_1 = (m_0, m_1)$  and  $x_2 = (m'_0, m'_1)$ , where  $m_0, m_1, m'_0, m'_1 \in \{0, 1\}^\lambda$ , 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  $i \in [n] \setminus \{1, 2, n\}$ ).  $f_{\text{ot}}$  allows everyone to learn  $(m_c, m'_c)$ .

Towards a contradiction, we assume a PrivP2P-BC, IA, CRS protocol  $\Pi$  with  $n \leq 3t + t_m$  securely computing  $f_{\text{ot}}$  exists, whose first-round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and second-round communication is over synchronous broadcast and peer-to-peer channels.

This proof is similar to the IA impossibility in [DRSY23], whose main proof idea for  $n \leq 3t$  and the synchronous PrivP2P-BC setting is as follows. The adversary splits the set of honest parties into two disjoint groups  $S_0$  and  $S_1$  (of size at most  $t$ ) where  $S_0$  contains  $P_1$  and  $S_1$  contains  $P_2$ . On behalf of a corrupt party  $P_n$ , she sends first-round messages computed based on  $x_n = 0$  to  $S_0$  and  $x'_n = 1$  to  $S_1$ . This creates confusion among the honest parties who are unable to identify whom to blame (for instance, honest parties in  $S_0$  do not know whether  $P_n$  is corrupt or the parties in  $S_1$  are corrupt) and therefore must compute the output. Their proof shows how this confusion can be exploited by the adversary to learn both  $m_0$  and  $m'_1$  (which is not allowed as per the ideal computation of  $f$ , as an adversary who corrupts neither  $P_1$  nor  $P_2$  would either learn  $(m_0, m'_0)$  or  $(m_1, m'_1)$ ).

We observe that the same proof idea as above can be used for our setting, with the difference being that our first round communication is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and we have an additional set of  $t_m$  honest parties, say  $S_m$ . If the message scheduling is such that the first-round

messages from  $P_n$  are not delivered to parties in  $S_m$ , then the same confusion (as in the above argument) can be created and the proof follows similarly. This completes the high-level description; we elaborate below.

Let  $S_0$ ,  $S_1$ ,  $S_m$  and  $\mathcal{I}$  denote disjoint sets of parties, where  $|S_0| = |S_1| = \frac{n-t-t_m}{2} \leq t$ ,  $|S_m| = t_m$  and  $|\mathcal{I}| = t$ . Suppose  $P_n \in \mathcal{I}$ ,  $P_1 \in S_0$  and  $P_2 \in S_1$ . Consider an execution of  $\Pi$  and the following scenarios:

**Scenario 1:** Suppose the adversary corrupts the  $t$  parties in  $\mathcal{I}$  that includes  $P_n$ . The adversary does the following:

**Round 1.** Compute and send, on behalf of the party  $P_n$ , messages based on input  $x_n = 0$  and  $x_n = 1$  to parties in  $S_0$  and  $S_1$  respectively. Behave honestly on behalf of other corrupt parties. Further, schedule the first-round messages such that the first-round message of  $P_n$  is *not delivered* to parties in  $S_m$ .

**Round 2.** Compute and send, on behalf of the corrupt parties, messages as if  $P_n$  computed her first-round message based on  $x_n = 0$ .

**Scenario 2:** Consider an adversary who corrupts parties in  $S_1$  (where  $S_1$  is as defined in Scenario 1 and in particular is such that it contains neither  $P_n$  nor  $P_1$ ). Suppose the input of honest  $P_n$  is  $x_n = 0$ . The adversary behaves as follows on behalf of corrupt parties:

**Round 1.** Behave honestly as per protocol specifications. Further, schedule the messages such that the first-round messages of  $P_n$  are not delivered to parties in  $S_m$ .

**Round 2.** Pretend (on behalf of the corrupt parties) to have received first-round messages from  $P_n$  based on  $x_n = 1$ . Note that the adversary can do this without being caught, since the setup is public (i.e. no private setup information is required to compute the first round message on behalf of honest  $P_n$  with respect to chosen input).

The first observation is that the view of honest parties in  $S_0$  in Scenario 1 is identically distributed to her view in Scenario 2. Since the identity of the corrupt parties is different in the two scenarios, it cannot be the case that the honest parties in  $S_0$  determines the identity of the cheater based on such a view (else, an honest person will be identified as the cheater in one of the two scenarios). We can thus conclude that the protocol execution must terminate with the honest parties obtaining an output.

The output obtained by honest parties (including  $P_n$ ) in Scenario 2 must consist of  $m_0$  as it should be computed with respect to the input  $x_n = 0$  of honest  $P_n$  and input  $(m_0, m_1)$  of honest  $P_1$ . Since the adversary's view in Scenario 1 subsumes the view of honest  $P_n$  in Scenario 2, we can conclude that the adversary in Scenario 1 is able to learn the output  $m_0$ .

Similarly, we can argue that if the adversary in Scenario 1 had computed the second-round messages of parties in  $\mathcal{I}$  based on  $x_n = 1$ , then she can obtain  $m'_1$ . This is because in such a case, the honest parties in  $S_1$  cannot identify whether  $P_n$  is corrupt or the  $t$  parties in  $S_0$  are pretending to have received first round message from  $P_n$  based on  $x_n = 0$  (say the latter is referred to as Scenario

2', whose output must include  $m'_1$  as it is based on honest inputs  $x_n = 1$  and  $x_2 = (m'_0, m'_1)$ . Therefore, by locally computing Round 2 messages on behalf of parties in  $\mathcal{I}$  based on  $x_n = 1$ , the adversary can obtain a view that subsumes the view of honest  $P_n$  in Scenario 2', enabling her to learn  $m'_1$  as well. This shows that there exists a strategy that allows the adversary corrupting  $\mathcal{I}$  to learn both  $m_0$  and  $m'_1$  which contradicts the security of  $\Pi$ ; completing the proof.

**Without PKI, Without Private Channels** In this section, we assume that the first-round peer-to-peer channels are public. We show that selective abort is impossible to achieve when  $n \leq t + t_d + t_m$ . This proof follows the approach of the previous results, where we describe an adversarial strategy and scheduling that reduces the argument to Cor 3.

**Theorem 6 (PubP2P-BC, SA, CRS,  $n \leq t + t_d + t_m$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with selective abort if  $n \leq t + t_d + t_m$ ; where the first round communication is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is over synchronous broadcast and peer-to-peer channels.*

*Proof.* We use the same function  $f_{\text{mot}}$  used in the proof of Theorem 1. Towards a contradiction, we assume a PubP2P-BC, SA, CRS protocol  $\Pi$  with  $n \leq t + t_d + t_m$  securely computing  $f_{\text{mot}}$  exists, whose first-round communication is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and second-round communication is over synchronous broadcast and peer-to-peer channels.

Consider a partition of the parties into three disjoint sets  $A, B$  and  $C$ , where  $|A| = t_m$ ,  $|B| = t_d$ , and  $|C| = t$ , respectively. Consider an execution of  $\Pi$  where the adversary *passively* corrupts the  $t$  parties in  $C$  and schedules the messages in the first round so that no messages from  $B$  are delivered to  $A$ .

The correctness of  $\Pi$  (which must hold as everyone including the passively controlled parties behaved honestly) dictates that this execution must result in output computation. Note that the second-round messages of parties in  $A$  do not depend on the first-round messages of parties in  $B$  (as parties in  $A$  did not receive any first-round messages from parties in  $B$ ). It is now easy to see that all the conditions of Cor 3 are satisfied. It now directly follows from Cor 3 that  $\Pi$  cannot securely compute  $f_{\text{mot}}$ , completing the proof.

### 3.2 Upper Bounds

**With PKI** In this section, we present two upper bounds for the setting where a PKI is available, the first-round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second-round communication is over synchronous peer-to-peer and broadcast channels.

An important tool in our constructions is one-or-nothing secret sharing, introduced in Damgård *et al.* [DMR<sup>+</sup>21], which we describe briefly below.

*One-or-Nothing Secret Sharing (1or0)*. One-or-nothing secret sharing is a special kind of secret sharing that allows a dealer to share a vector of secrets. As the name suggests, at most one among these secrets is eventually reconstructed. Once the shares are distributed, each receiver votes on the index of the value to reconstruct by producing a “ballot”. If the receiver is unsure which index to vote for, she can publish a special equivocation ballot instead. Damgård *et al.* define a non-interactive variant of such a secret sharing, which supports parties being able to vote even if they have not received the shares. We recall the syntax in Appendix B.1. Informally, the properties required from a one-or-nothing secret sharing scheme are as follows.

**$\delta$ -Correctness.** This property requires that if at least  $\delta$  parties produce their ballot using the same index  $v$  (and the rest produce their ballot with  $\perp$  i.e. the special equivocation ballot), then the secret at index  $v$  is reconstructed.

**Privacy.** If no honest party produced their ballot using  $v$ , then the adversary learns nothing about the secret at index  $v$ .

**Contradiction-privacy.** If two different honest parties produce their ballots using different votes (i.e. vote for different indices), then the adversary learns nothing at all.

While the definition in Damgård *et al.* defined correctness with respect to  $\delta = n - t$ , we consider a more general version of the same to adapt it to our setting with  $(t_d, t_m)$ -asynchrony. Their work presents a construction of a non-interactive one-or-nothing secret sharing scheme with  $(n - t)$ -correctness when  $n > 2t$ . We observe that, more generally, the same construction (with a minor tweak) serves as a non-interactive one-or-nothing secret sharing scheme with  $\delta$ -correctness when  $\delta > t$  holds. We defer the formal details of this construction to Appendix B.2.

Looking ahead, in our upper bounds, we use a non-interactive one-or-nothing secret sharing scheme with (1)  $(n - t - t_m)$ -correctness when  $n > 2t + t_m$ , and (2)  $(n - t_m)$ -correctness when  $n > t + t_m$ .

*Protocol Overview of P2P-BC, ID, PKI,  $2t < n$  [DMR<sup>+</sup>21]*. Since both our upper bounds are constructed by slight modifications to the fully synchronous P2P-BC, ID, PKI,  $2t < n$  construction of Damgård *et al.*, we give an overview of their construction below (we refer to [DMR<sup>+</sup>21] for the formal description). Their work presents a compiler that transforms a fully synchronous BC-BC, ID, PKI protocol  $\Pi_{bc}$  to a fully synchronous P2P-BC, ID, PKI,  $2t < n$  protocol  $\Pi_{p2pbc}$ . The following steps are executed in  $\Pi_{p2pbc}$ : In the first round over synchronous peer-to-peer channels, the parties send their first-round message of  $\Pi_{bc}$  along with a signature to each of their peers. In the second round (over broadcast), the parties do the following:

1. Compute and broadcast a garbling of their next-message function (that has hardcoded input and randomness of a party, takes as input the first-round messages of  $\Pi_{bc}$  from all parties and computes the second-round message of the party according to  $\Pi_{bc}$ );

2. Use the non-interactive **1or0** scheme with  $(n - t)$ -correctness to share all the input labels for their garbled circuit;
3. Based on the first-round messages received, “vote”<sup>7</sup> for which labels to reconstruct corresponding to everyone’s garbled circuit (vote for  $\perp$  in case no or invalid first-round message was received);
4. Compute a zero-knowledge proof to prove correctness of the actions taken in the second round; and
5. Echo all the first-round messages of  $\Pi_{bc}$  with the corresponding signatures received from the other parties in the first round.

During output computation, parties first verify the zero-knowledge proofs and signatures and catch the relevant party in case anyone’s proof fails or there exist distinct first-round messages with valid signatures from the same party. They then proceed to reconstruct the appropriate labels of the garbled circuits of all parties. A party  $P_i$  is blamed if the reconstruction of the label corresponding to her first-round message fails. If all the labels are reconstructed successfully, the parties proceed to evaluating the garbled circuits and obtain the second round messages of  $\Pi_{bc}$  for all parties; which is subsequently used to obtain the output.

Intuitively, the protocol achieves identifiable abort when  $n > 2t$ . This is because, to avoid being caught, a corrupt party needs to send her 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 and she would be implicated). Further, she cannot afford to send different first-round messages to different honest parties with valid signatures (otherwise the contradictory signatures would implicate her). Contradiction privacy of one-or-nothing secret sharing ensures that in such a case the adversary does not learn any of the labels. Next, the zero-knowledge proof in the second round ensures that every corrupt party garbles and shares its garbled circuit labels correctly. Lastly, if no party is caught, it must hold each party’s first-round message is echoed by at least  $(n - t)$  parties who would vote accordingly. The  $(n - t)$ -correctness of one-or-nothing secret sharing ensures that in such a case, exactly one label from each label pair is reconstructed, which enables the underlying protocol  $\Pi_{bc}$  to be carried out.

We are now ready to present the upper bounds for our setting where the first-round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony.

**Theorem 7 (P2P-BC, IA, PKI,  $2t + t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and let  $2t + t_m < n$ . Let  $\Pi_{bc}$  be a BC-BC, ID, PKI protocol that securely computes  $f$  with the additional constraint that the straight-line simulator can extract inputs from corrupt parties’ first-round messages. Assume the existence of a secure garbling scheme, digital signature scheme, non-interactive one-or-nothing secret sharing scheme [DMR<sup>+</sup>21], non-interactive key agreement scheme and non-interactive zero-knowledge proof system. Then, there exists a P2P-BC, ID, PKI protocol that securely computes  $f$  over two rounds, the first of which is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over a synchronous broadcast and peer-to-peer channel.*

<sup>7</sup>Note that the one-or-nothing secret sharing is non-interactive; thereby “share” and “vote” can be executed in the same round.

*Proof (Sketch).* We sketch a two-round construction  $\Pi_{\text{asynpcp2pbc}}^{\text{id-abort}}$  achieving identifiable abort when  $2t + t_m < n$ , where the first round uses peer-to-peer communication with  $(t_d, t_m)$ -asynchrony, and the second round uses synchronous broadcast and peer-to-peer communication. This construction is the same as the construction of [DMR<sup>+</sup>21] described above, except that we use the non-interactive one-or-nothing secret sharing scheme satisfying  $(n - t - t_m)$ -correctness when  $n > 2t + t_m$ .

The main difference is that in our setting it may so happen that an honest party's first round message is echoed by only  $n - t - t_m$  parties (because her messages to up to  $t_m$  honest parties may be dropped, and  $t$  corrupt parties may pretend to have not received her first-round message sent over peer-to-peer channels). Therefore, we can implicate a party as being a cheater only if her message is echoed by fewer than  $n - t - t_m$  parties. Accordingly, to ensure that the appropriate labels (corresponding to the first-round message) are successfully reconstructed when no cheater is identified, we use the non-interactive one-or-nothing secret sharing with  $(n - t - t_m)$ -correctness.

At a high level, the proof that the protocol  $\Pi_{\text{asynpcp2pbc}}^{\text{id-abort}}$  achieves identifiable abort is similar to the one shown by Damgård *et al.* [DMR<sup>+</sup>21]. First, if a party sent two different first-round messages with valid signatures, then this party can be easily implicated. Second, if a corrupt party is not caught, then we can infer that at least one honest party received a first-round message with a valid signature. This is because a party can avoid being implicated as a cheater only if at least  $n - t - t_m > t$  parties echo her first round message. Finally, by the simulation-soundness of the NIZK we ensure that every corrupt party garbles and shares its garbled circuit labels correctly. Note that if no party is caught, the  $(n - t - t_m)$ -correctness property of the one-or-nothing secret sharing scheme ensures that one label corresponding to each wire of all the garbled circuits is successfully reconstructed, and therefore the protocol  $\Pi_{\text{bc}}$  can be executed.

**Theorem 8 (P2P-BC, UA, PKI,  $t + t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and let  $t + t_m < n$ . Let  $\Pi_{\text{bc}}$  be a BC-BC, UA, PKI protocol that securely computes  $f$  with the additional constraint that the straight-line simulator can extract inputs from corrupt parties' first-round messages. Assume the existence of a secure garbling scheme, non-interactive one-or-nothing secret sharing scheme [DMR<sup>+</sup>21], non-interactive key agreement scheme and non-interactive zero-knowledge proof system. Then, there exists a P2P-BC, UA, PKI protocol that securely computes  $f$  over two rounds, the first of which is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over synchronous broadcast and peer-to-peer channels.*

*Proof (Sketch).* In order to construct a protocol  $\Pi_{\text{asynpcp2pbc}}^{\text{un-abort}}$  in the setting of Theorem 8, we make the following modifications to the protocol  $\Pi_{\text{p2pbc}}$  of [DMR<sup>+</sup>21] described above.

- First, since in this case we are not required to identify a cheater (as the goal is unanimous abort), the protocol simply outputs  $\perp$  in all the cases where

the cheater was identified. Therefore, we can avoid the use of signatures, which are needed only for cheater identification.

- Second and more crucial, we use the non-interactive one-or-nothing secret sharing scheme with  $(n - t_m)$ -correctness when  $n > t + t_m$ . This ensures that when everyone behaves honestly and it so happens that a party's first round message is received by only  $n - t_m$  parties (who would all vote for the same value and others vote for  $\perp$ ), the appropriate labels of all the garbled circuits corresponding to this party's first-round message are successfully reconstructed and subsequently the protocol results in output computation.

We now briefly analyze the security of  $\Pi_{\text{asyn}cp2\text{pb}c}^{\text{un-abort}}$ . At a high-level if the adversary in the first round sends two different first-round messages of  $\Pi_{bc}$  to a pair of honest parties, then these parties will vote for different values (at least for the bits where the two messages differ). However, we are guaranteed from the contradiction privacy of the one-or-nothing secret sharing scheme that no information about the labels for the corresponding wires in the garbled circuit will be revealed. Since the parties cannot proceed to evaluating the garbled circuit, all honest parties abort unanimously. If this is not the case, then there are two possibilities: (a) no honest party received a valid first-round message from the adversary; or (b) there is a unique valid first-round message of  $\Pi_{bc}$  that the adversary sent to a subset of the honest parties (others received nothing or an invalid message). In case (a) the privacy of the one-or-nothing secret sharing scheme ensures that no information is revealed on any of the labels and all parties unanimously abort. In case (b), depending on whether at least  $n - t_m$  parties echo the same first-round message or not, the protocol will either result in successful reconstruction of the label (corresponding to the unique first-round message) or result in parties aborting unanimously. Lastly, note that the adversary learns at most one label per wire of an honest party's garbled circuit, due to the contradiction privacy of the one-or-nothing secret sharing scheme. Since the second-round communication only involves broadcast communication, unanimity amongst the honest parties is easy to maintain.

**Without PKI, Without Private Channels, from One-or-Nothing Secret Sharing with Intermediaries** In this section, we present two constructions for the CRS setting where the first-round communication is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is over synchronous peer-to-peer and broadcast channels. In contrast to the obfuscation-based constructions in Section 3.2, the ones in this section can be built from standard assumptions and for any polynomial number of parties. However, the bounds on  $t, t_d$  and  $t_m$  they achieve are looser.

Next, we analyze the following constraints with respect to the above parameters.

**(C.1)**  $\epsilon + t \leq \beta$  must hold. To see this suppose  $\epsilon + t > \beta$  and  $(\epsilon - 1)$  honest parties produce their ballot for the same value  $v$ . Then, those ballots and the  $t$  additional ballots on  $v$  (that the adversary can produce) are sufficient

to reconstruct the secret at  $v$ , based on the property of  $\beta$ -correctness (as  $\epsilon - 1 + t \geq \beta$  would hold). However, this contradicts  $\epsilon$ -privacy.

**(C.2)**  $2\epsilon > n - t$  ensures that the secret at at most one index is reconstructed, even when honest parties vote for different indices. This is because if this condition holds, then it cannot be the case that a set of  $\epsilon$  honest parties vote for  $v$  (necessary to reconstruct the secret at index  $v$  based on privacy) and a different (disjoint) set of  $\epsilon$  honest parties vote for  $v'$  (necessary to reconstruct the secret at index  $v'$  based on privacy). Looking ahead, this property is useful for our IA upper bound construction.

From the above two constraints, we get that  $\frac{n-t}{2} < \epsilon \leq \beta - t$ , which implies that  $\beta > \frac{n+t}{2}$  holds. Note that the construction of Damgård *et al.* [DRSY23] with  $\epsilon = n - 2t$  and  $\beta = n - t$  satisfies both these constraints as it works for  $n > 3t$ . We give a brief overview of this construction below, and then explain how it can be adapted to our setting.

*Overview of 1or0wi Construction [DRSY23].* At a high-level, in this construction, the dealer computes two levels of threshold secret sharing (Section A.2) for each of the secrets (among the vector of secrets). First, she shares a secret, say  $s$ , using threshold  $(n - t - 1)$ , creating for each party  $R$  a share  $s_R$ . Next, each of these first-level shares is re-shared using threshold  $(n - 2t - 1)$ .

All the parties now act as intermediaries to pass on subshares of  $s_R$  to  $R$ . This is done using a tool referred to as *transferrable encryption*. Briefly, this works as follows: For the sub-share of  $s_R$  intended for intermediary  $I$ , the dealer broadcasts an encryption  $c$  of the sub-share under a public key received from  $I$ . Simultaneously,  $I$  prepares an encryption, say  $c'$  of its (one-time) secret key (corresponding to the public key using which the dealer prepared this encryption  $c$ ) using the public key received from  $R$ . This encryption  $c'$  is referred to as a transfer key (as it transfers the ability to decrypt  $c$  to  $R$ ). Depending on which value should be decrypted (corresponding to which secret  $R$  wishes to reconstruct),  $R$  broadcasts the relevant decryption key which can be used to decrypt  $c'$  to retrieve the secret key of  $I$ , which is subsequently used to decrypt  $c$  to obtain the intended sub-share. All the actions are augmented with NIZK proofs to ensure correctness.

It is important to note that the public keys may not have been exchanged consistently, but there are enough intermediaries to reconstruct successfully when no one is identified as corrupt. More specifically, a party is identified as corrupt if she is in conflict with more than  $t$  parties with respect to their public keys (this ensures that an honest party is never implicated). Therefore, when no one is identified, it must be the case that the dealer is in agreement with at least  $(n - t)$  intermediaries about their public keys. Furthermore, each  $R$  must be in agreement with at least  $(n - t)$  intermediaries about its public key. Therefore, there are at least  $(n - 2t)$  intermediaries who are in agreement with both the dealer and  $R$ , which is sufficient to enable the transfer of  $s_R$  to  $R$  (as  $s_R$  is threshold shared using threshold  $(n - 2t - 1)$ ). It is crucial that  $(n - 2t) > t$  holds, as otherwise the  $t$  corrupt parties acting as intermediaries would be sufficient to re-

cover the share  $s_R$  meant for honest R. This choice of thresholds leads Damgård *et al.* [DRSY23] to obtain  $(n - t)$ -identifiability and  $(n - 2t)$ -privacy (as per constraint **(C.1)**). Moreover, it follows from **(C.2)** that  $2(n - 2t) > n - t$ , i.e.  $n > 3t$ , must hold so that the secret at at most one index is reconstructed.

*Adapting to  $(t_d, t_m)$ -Asynchrony With Identifiability.* To adapt the above to the case when the public keys are communicated over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, we observe that an honest dealer must have received public keys from at least  $(n - t_d)$  intermediaries among which at most  $t$  may not be in agreement (as they may be corrupt). So, there are at least  $(n - t_d - t)$  intermediaries who will be in agreement with the dealer (with respect to their public keys which the dealer used to compute the  $c$  ciphertexts). If not, the dealer can be implicated as corrupt. Similarly, we observe that R’s public key would be received by at least  $(n - t_m)$  intermediaries among which at most  $t$  will not be in agreement. So, there are at least  $(n - t_m - t)$  intermediaries who will be in agreement with R (with respect to R’s public keys that the intermediaries used to compute the  $c'$  ciphertexts). If not, R can be implicated as corrupt. Thus, there are at least  $(n - 2t - t_d - t_m)$  intermediaries who are in agreement with the dealer (with respect to the intermediary’s public key) and R (with respect to R’s public key). Setting the second-level threshold to  $(n - 2t - t_d - t_m - 1)$  ensures that this set of intermediaries is enough to reconstruct  $s_R$ . To ensure that a set of  $t$  corrupt intermediaries are not sufficient to recover the share  $s_R$  meant for honest R,  $(n - 2t - t_d - t_m) > t$  i.e.  $n > 3t + t_m + t_d$  must hold. Tying up loose ends, we choose the first-level threshold to be  $(n - t_m - t - 1)$  to achieve  $(n - t_m - t)$ -identifiability as per the requirement in our IA MPC protocol. We thus extend the construction to a one-or-nothing secret sharing with intermediaries with  $(n - t_m - t)$ -identifiability (and  $(n - t_m - t)$ -correctness) and  $(n - 2t - t_m)$ -privacy (respecting constraint **(C.1)** mentioned above). Furthermore, it follows from **(C.2)** that if  $2(n - 2t - t_m) > n - t$  i.e.  $n > 3t + 2t_m$  holds, then the secret at at most one index can be successfully reconstructed.

Based on the above observations and minor modifications to the construction of `1or0wi` of Damgård *et al.* [DRSY23], we obtain the following (details appear in Section C.2).

**Theorem 9 ([DRSY23] (with minor modifications)).** *Assume the existence of a public key encryption scheme with CPA security, and a secure non-interactive zero knowledge proof system. Then, there exists a maliciously secure one-or-nothing secret sharing with intermediaries with  $(n - t_m - t)$ -identifiability (which implies  $(n - t_m - t)$ -correctness) and  $(n - 2t - t_m)$ -privacy when  $n > 3t + t_m + t_d$  and  $n > 3t + 2t_m$  holds.*

Next, we adapt the construction of `1or0wi` to the asynchronous setting where identifiability is not required.

*Adapting to  $(t_d, t_m)$ -Asynchrony Without Identifiability.* We again consider the case when the public keys are communicated over peer-to-peer channels with

$(t_d, t_m)$ -asynchrony but demand only correctness (not identifiability). Suppose everyone behaves honestly. Then, the dealer must have received public keys from at least  $(n - t_d)$  intermediaries who will be in agreement with the dealer (with respect to their public keys that the dealer used to compute the  $c$  ciphertexts). Similarly, we observe that R's public key would be received by at least  $(n - t_m)$  intermediaries who will be in agreement with R (with respect to R's public keys that the intermediaries used to compute the  $c'$  ciphertexts). If these conditions are not satisfied, it must be the case that some party misbehaved and all parties simply abort. When everyone behaves honestly, we can conclude that there are at least  $(n - t_d - t_m)$  intermediaries who are in agreement with both the dealer and R. Setting the second-level threshold to  $(n - t_d - t_m - 1)$  ensures that this set of intermediaries is enough to reconstruct  $s_R$ . To ensure that a set of  $t$  corrupt intermediaries are not sufficient to recover the share  $s_R$  of an honest R,  $(n - t_d - t_m) > t$  i.e.  $n > t + t_m + t_d$  must hold. For use in our UA MPC protocol, we obtain a construction of one-or-nothing secret sharing with intermediaries with  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy (respecting constraint **(C.1)** mentioned above). Further, it follows from **(C.2)** that if  $2(n - t - t_m) > n - t$  i.e.  $n > t + 2t_m$  holds, then the secret at at most one index can be successfully reconstructed.

We thus obtain the following (details appear in Section C.2).

**Theorem 10 ([DRSY23] (with minor modifications)).** *Assume the existence of a public key encryption scheme with CPA security, and a secure non-interactive zero knowledge proof system. Then, there exists a maliciously secure one-or-nothing secret sharing with intermediaries with  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy when  $n > t + t_m + t_d$  and  $n > t + 2t_m$  holds.*

We are now ready to present our upper bounds. Both our upper bounds are constructed via slight modifications to the fully synchronous PubP2P-BC, ID, CRS,  $3t < n$  construction of Damgård *et al.* Their work presents a compiler that transforms a two-broadcast round ID, CRS protocol  $\Pi_{bc}$  to a fully synchronous PubP2P-BC, ID, CRS,  $3t < n$  protocol  $\Pi_{p2pbc}$  where the peer-to-peer channels are assumed to be public. Roughly speaking, their compiler proceeds similarly to the compiler of Damgård *et al.* [DMR<sup>+</sup>21] (described in Section 3.2) which involves parties computing garbled circuits corresponding to the next message function of  $\Pi_{bc}$  and secret sharing the labels of these garbled circuits. A crucial difference is that the tool used for secret sharing in the CRS setting is the maliciously-secure one-or-nothing secret sharing with intermediaries.

We observe that plugging in the (malicious secure) one-or-nothing secret sharing with intermediaries with modified parameters of correctness, privacy and identifiability in the above compiler yields P2P-BC IA and UA upper bounds tolerating  $(t_d, t_m)$ -asynchrony in the first round for certain range of thresholds. We state the formal theorems below.

**Theorem 11 (PubP2P-BC, IA, CRS,  $3t + t_m + t_d < n$ ,  $3t + 2t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and suppose  $3t + t_m + t_d < n$ ,  $3t + 2t_m < n$  holds. Let  $\Pi_{bc}$  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. Assume the existence of a secure garbling scheme and a maliciously-secure one-or-nothing secret sharing with intermediaries with  $(n - t_m - t)$ -identifiability and  $(n - 2t - t_m)$ -privacy. Then, there exists a PubP2P-BC, ID, CRS protocol that securely computes  $f$  over two rounds, the first of which is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over a synchronous broadcast and peer-to-peer channel.

*Proof (Sketch).* We observe that when the one-or-nothing secret sharing with intermediaries with  $(n - t_m - t)$ -identifiability and  $(n - 2t - t_m)$ -privacy is plugged into the compiler of Damgård *et al.* [DRSY23], it yields a PubP2P-BC, ID, CRS protocol where the first round communication can be carried over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is executed over a synchronous broadcast and peer-to-peer channel.

The main difference is that in the asynchronous setting it may so happen that an honest party's first round message is echoed by only  $n - t_m - t$  parties (because her messages may be dropped to  $t_m$  honest parties, and  $t$  corrupt parties may pretend to have not received her first-round message sent over peer-to-peer channels). Therefore, we can implicate a party as being a cheater only if her message is echoed by fewer than  $n - t_m - t$  parties. The property of  $(n - t_m - t)$ -identifiability ensures that in such a case (when a party's first-round message is supported by  $n - t_m - t$  parties), either the appropriate labels (corresponding to a party first-round message) are successfully reconstructed or a cheater is identified; maintaining the IA guarantee.

The above also means that on behalf a corrupt party, the adversary needs to send a consistent first round message to only  $n - t_m - 2t$  honest parties. This is because their support combined with the additional support of  $t$  corrupt parties' would be sufficient to avoid being implicated and obtain the labels corresponding to this first-round message. If the adversary has this behaviour, then  $(n - 2t - t_m)$ -privacy ensures that the adversary gets at most one label corresponding to her first round message based on the analysis of constraint **(C.2)** above – In more detail, in order for two different labels to be successfully reconstructed, the adversary must send different first-round messages to two disjoint sets of honest parties, each of size at least  $n - t_m - 2t$ . However, this is not possible since  $2(n - t_m - 2t) \leq n - t$  contradicts our assumption that  $3t + 2t_m < n$ .

Lastly, we note that the formal proof of security would follow identically to that of Damgård *et al.* [DRSY23], except that the privacy and identifiability properties are invoked with respect to the one-or-nothing secret sharing with intermediaries with the modified parameters of  $(n - t_m - t)$ -identifiability and  $(n - 2t - t_m)$ -privacy.

**Theorem 12 (PubP2P-BC, UA, CRS,  $t + t_m + t_d < n$ ,  $t + 2t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and suppose  $t + t_m + t_d < n$ ,  $t + 2t_m < n$  holds. Let  $\Pi_{bc}$  be a BC-BC, UA, CRS protocol that securely computes  $f$  with the additional constraint that the straight-line simulator can*

extract inputs from corrupt parties' first-round messages. Assume the existence of a secure garbling scheme and a maliciously-secure one-or-nothing secret sharing with intermediaries with  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy. Then, there exists a PubP2P-BC, UA, CRS protocol that securely computes  $f$  over two rounds, the first of which is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over a synchronous broadcast and peer-to-peer channel.

*Proof (Sketch).* We observe that the following minor modifications to the compiler of Damgård *et al.* [DRSY23] yields a PubP2P-BC, UA, CRS protocol where the first round communication can be carried over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony and the second round communication is executed over a synchronous broadcast and peer-to-peer channel.

- First, since in this case we are not required to identify a cheater (as the goal is unanimous abort), the protocol simply outputs  $\perp$  in all the cases where the cheater was identified.
- Second and more crucial, we use the one-or-nothing secret sharing with intermediaries scheme with  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy when  $n > t + t_m + t_d$  and  $n > t + 2t_m$  holds.

The main difference is that in the asynchronous setting it may so happen that a party's first round message is echoed by only  $n - t_m$  parties in an all-honest execution (because her messages to up to  $t_m$  honest parties may be dropped). The property of  $(n - t_m)$ -correctness ensures that the appropriate labels of all the garbled circuits corresponding to this party's first-round message are successfully reconstructed and subsequently the protocol results in output computation. Next, since the constraint **(C.2)** i.e.  $n - t < 2\epsilon$  is satisfied (since  $\epsilon = n - t - t_m$  and we assume  $n > t + 2t_m$ ), we can infer that the adversary who sends inconsistent first-round messages to different sets of honest parties obtains at most one label corresponding to her first round message (therefore security of garbled circuits can be invoked to argue security of the protocol).

The formal proof of security would follow identically to the compiler of [DRSY23], except that the privacy and correctness properties are invoked with respect to the one-or-nothing secret sharing with intermediaries with the modified parameters of  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy.

**Without PKI, With private channels, from One-or-Nothing Secret Sharing with Intermediaries** The PubP2P-BC CRS upper bounds of Theorem 11 and Theorem 12 work in the setting where the first round peer-to-peer communication is public (similar to the upper bounds in [DRSY23]). While in the fully synchronous setting, availability of private channels in the first round does not seem to be useful in obtaining stronger security guarantees or achieving feasibility for wider threshold ranges (except the case where corruption threshold is one), we observe that this makes a difference when we adapt the approach based on one-or-nothing secret sharing with intermediaries to the setting where first-round communication is over peer-to-peer channels with  $(t_d, t_m)$ -asynchrony.

As described in Section 3.2, recall that in the construction of `1or0wi`, the set of intermediaries that enabled the share transfer (via the transferrable encryption) belonged to the non-deaf set of the dealer (i.e. belonged to the set of parties from whom the dealer received the public keys in the first round) as well as belonged to the non-mute set of the receiver `R` (i.e. belonged to the set of parties who received the public key from `R` in the first round).

Transferrable encryption enabled this share transfer to occur non-interactively over broadcast channel, as the dealer could “share” (by preparing encryptions to the intermediaries) and the receivers could “vote” simultaneously. When this is plugged in the `PubP2P-BC` MPC protocols, the only communication required in the first round with respect to `1or0wi` was exchanging public keys which could be done over public peer-to-peer channels.

We observe that if private peer-to-peer channels were available in the first round, then the sharing could be split into two parts – In the first round, the sub-shares could be handed over to the intermediaries privately in the first round itself. In the second round, over broadcast, the intermediaries could publish the transfer keys and the receivers could vote simultaneously (same as in the original construction). Note that with this modification the set of intermediaries that enable the share transfer is changed. More specifically, the intermediaries that enable the share transfer are those who belonged to the non-mute set of the dealer (i.e. received the private shares from the dealer in the first round) as well as belonged to the non-mute set of the receiver `R` (i.e. belonged to the set of parties who received the public key from `R` in the first round). Following the analysis similar to Section 3.2, we observe that this modification in the `1or0wi` (of communicating the sub-shares to the intermediaries privately in the first round) results in the condition  $(n - t - t_m - t - t_m) > t$  i.e.  $n > 3t + 2t_m$  for the case with identifiability and the condition  $(n - t_m - t_m) > t$  i.e.  $n > t + 2t_m$  without identifiability. We thus conclude that using private channels yields a better feasibility bound (with no dependence on  $t_d$ ) for `1or0wi`, which in turn improves the feasibility bounds of the MPC constructions. We state the inferences below.

**Observation 1 (PrivP2P-BC, IA, CRS,  $3t + 2t_m < n$ )** *Let  $f$  be an efficiently computable  $n$ -party function and suppose  $3t + 2t_m < n$  holds. Assume the existence of a maliciously-secure one-or-nothing secret sharing with intermediaries with  $(n - t_m - t)$ -identifiability and  $(n - 2t - t_m)$ -privacy. Then, there exists a `PrivP2P-BC`, `ID`, `CRS` protocol that securely computes  $f$  over two rounds, the first of which is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over a synchronous broadcast and peer-to-peer channel.*

**Observation 2 (PrivP2P-BC, UA, CRS,  $t + 2t_m < n$ )** *Let  $f$  be an efficiently computable  $n$ -party function and suppose  $t + 2t_m < n$  holds. Assume the existence of a maliciously-secure one-or-nothing secret sharing with intermediaries with  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy. Then, there exists a `PrivP2P-BC`, `UA`, `CRS` protocol that securely computes  $f$  over two rounds, the first of which is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over a synchronous broadcast and peer-to-peer channel.*

**Without PKI, from Obfuscation** In this section, we discuss the constructions from diO. These constructions have the purpose of showing feasibility, or rather, to give an evidence of *infeasibility* of a negative result in those settings. We provide an overview of the constructions below.

Informally, differing-inputs obfuscation (diO, refer Appendix A.1) [BGI<sup>+</sup>01,ABG<sup>+</sup>13] is an algorithm that transforms a program into a form in which it can still be evaluated, but its inner workings are hidden. The security guarantee of diO is that, given two programs  $P_0$  and  $P_1$  such that it is computationally hard to find an input  $x$  such that  $P_0(x) \neq P_1(x)$ , it is also hard to distinguish between  $\text{diO}(P_0)$  and  $\text{diO}(P_1)$ .<sup>8</sup>

Our constructions use something we call *puncturable public-key encryption*. This new type of encryption allows us to create *punctured* decryption keys that are unable to decrypt messages from certain senders, while behaving correctly with respect to all other senders.

#### *Puncturable Public-Key Encryption*

*Definitions* Puncturable public key encryption (PPKE) requires the use of the sender’s secret key — in addition to the receiver’s public key — to encrypt. In a PPKE scheme, the decryption key can be punctured with respect to a set of sender public keys in such a way that ciphertexts produced by those senders do not reveal the encrypted message even given the receiver’s punctured decryption key. More formally, a PPKE scheme consists of four algorithms:

$\text{keygen}_S(1^\lambda) \rightarrow (\text{pk}_S, \text{sk}_S)$  generates a key pair belonging to a sender.  
 $\text{keygen}_R(1^\lambda) \rightarrow (\text{pk}_R, \text{sk}_R)$  generates a key pair belonging to a receiver.  
 $\text{pkeygen}_R(1^\lambda, \mathcal{P}) \rightarrow (\text{pk}_R, \text{sk}_R)$  generates a key pair belonging to a receiver, where the decryption key is punctured at the set of public keys  $\mathcal{P}$ .  
 $\text{enc}(\text{sk}_S, \text{pk}_S, \text{pk}_R, \text{msg}) \rightarrow c$  encrypts a message.  
 $\text{dec}(\text{sk}_R, \text{pk}_S, c) \rightarrow \text{msg}$  decrypts a ciphertext.

**Definition 4 (Secure PPKE).** *A PPKE scheme is secure if:*

1. *It is correct in the usual sense.*
2. *It is semantically secure in the usual sense.*
3. *It is securely puncturable, meaning that given a receiver key pair produced by  $\text{pkeygen}_R(1^\lambda, \mathcal{P})$ ,*
  - (a) *correctness holds for all sender public keys  $\text{pk}_S \notin \mathcal{P}$ , but*
  - (b) *semantic security holds for  $\text{pk}_S \in \mathcal{P}$  even given the secret key  $\text{sk}_R$ .*
4. *It is puncture hiding, meaning that the public key  $\text{pk}_R$  does not reveal whether it is punctured or not (or the points of puncture).*

---

<sup>8</sup>It has been conjectured that differing-inputs obfuscation does not exist [BSW16]. However, its non-existence has not been definitively proven; finding such definitive proof is a very hard open problem.

**Observation 3 (Inter-Key Non-Malleability)** *A given ciphertext  $c$  under  $\text{pk}_S$  cannot be transformed to  $c'$  under a different  $\text{pk}'_S$  such that the plaintexts  $\text{msg}$  recovered by decrypting  $c$  under  $\text{pk}_S$  and  $\text{msg}'$  recovered by decrypting  $c'$  under  $\text{pk}'_S$  are correlated.*

*In other words, an adversary who provides  $\text{msg}_0$  and  $\text{msg}_1$  and is shown  $c = \text{enc}(\text{sk}_S, \text{pk}_S, \text{pk}_R, \text{msg}_b)$  cannot come up with a ciphertext  $c'$  and  $\text{pk}'_S \neq \text{pk}_S$  such that, given  $\text{dec}(\text{sk}_R, \text{pk}'_S, c')$ , she can guess  $b$  with non-negligible advantage.*

*Proof.* We start with an honestly generated  $(\text{pk}_S, \text{sk}_S)$  and an honestly generated  $(\text{pk}_R, \text{sk}_R)$  punctured at  $\text{pk}_S$ . Given  $\text{msg}_0$  and  $\text{msg}_1$  provided by the adversary  $\mathcal{A}$  (who knows  $\text{pk}_S, \text{pk}_R$  and  $\text{sk}_R$ ), we choose  $b \leftarrow \{0, 1\}$ , and, encrypt  $c \leftarrow \text{enc}(\text{sk}_S, \text{pk}_S, \text{pk}_R, \text{msg}_b)$ , and give  $c$  to  $\mathcal{A}$ . If  $\mathcal{A}$  could transform  $c$  to  $c'$  under  $\text{pk}'_S$  (where  $\text{pk}_R$  is not punctured at  $\text{pk}'_S$ ) such that the decryption of  $c'$  is not independent of  $\text{msg}_b$ ,  $\mathcal{A}$  could then decrypt  $c'$  (using  $\text{sk}_R$ ) and guess  $b$ , which should be impossible by secure puncturability.

*Construction* We propose a simple PPKE scheme that uses diO, a pseudorandom function (PRF), and a puncturable PRF (PPRF). The receiver’s public encryption key is  $\text{pk}_R = \text{OP}^{\text{enc}}$ , which is an obfuscation of the encryption program  $\text{P}_{\mathbf{k}, \mathbf{k}'}^{\text{enc}}$  (with hardcoded PPRF key  $\mathbf{k}$  and PRF key  $\mathbf{k}'$ , which also form  $\text{sk}_R$ ). To generate a punctured public key, the key  $\mathbf{k}$  embedded in  $\text{P}_{\mathbf{k}, \mathbf{k}'}^{\text{enc}}$  is punctured.

The sender’s secret key is a signing key  $\text{sk}_S = \text{sk}$ , and the corresponding public key is the verification key  $\text{pk}_S = \text{vk}$ . The sender encrypts the message  $\text{msg}$  by choosing a random nonce  $\text{nonce}$ , signing  $(\text{msg}, \text{nonce}, \text{“encrypt”})$  to get  $\sigma$ , and evaluating  $\text{OP}^{\text{enc}}$  on  $(\text{vk}, \text{msg}, \text{nonce}, \sigma)$  to get  $c$ .<sup>9</sup>

```

 $\text{P}_{\mathbf{k}, \mathbf{k}'}^{\text{enc}}(\text{vk}, \text{msg}, \text{nonce}, \sigma) :$ 


---


if  $\text{SIG.verify}(\text{vk}, (\text{msg}, \text{nonce}, \text{“encrypt”}), \sigma) = \text{accept}$  then
   $c' := \text{PPRF}_{\mathbf{k}}(\text{vk}) \oplus \text{PRF}_{\mathbf{k}'}(\text{nonce}) \oplus \text{msg}$ 
   $c := (c', \text{nonce})$ 
  return  $c$ 
end if

```

The receiver uses  $\text{sk}_R = (\mathbf{k}, \mathbf{k}')$  to decrypt by computing  $\text{msg} = c' \oplus \text{PPRF}_{\mathbf{k}}(\text{vk}) \oplus \text{PRF}_{\mathbf{k}'}(\text{nonce})$ . (If  $\mathbf{k}$  is punctured at  $\text{vk}$ , a default output of  $\perp$  is given.)

By puncturing  $\mathbf{k}$  at a set of verification keys  $\mathcal{P}$ , we can ensure that ciphertexts produced by the owners of these verification keys will look uniformly random even given both  $\text{OP}^{\text{enc}}$  and the punctured key  $\mathbf{k}$ . Furthermore,  $\text{OP}^{\text{enc}}$  with a punctured  $\mathbf{k}$  will be computationally indistinguishable from  $\text{OP}^{\text{enc}}$  with an intact  $\mathbf{k}$ , since signatures are hard to forge, and it is thus hard for anyone other than the owner of the corresponding signing keys to find inputs on which the two obfuscated programs disagree.

<sup>9</sup>We use the same signature scheme for multiple purposes in our secure computation construction; to ensure that signatures produced for one purpose cannot be used to break security elsewhere, we always include a keyword (such as “encrypt”) in our signed messages.

*MPC From diO* Consider the function  $P_{dk,dk'}$ , in which the puncturable public-key decryption keys  $dk$  and  $dk'$  are hard-coded. The obfuscation  $OP = diO(P_{dk,dk'})$ , as well as the PPKE receiver keys  $ek$  and  $ek'$ , are available in a CRS. ( $dk'$  and  $ek'$  are necessary for a technicality in the proof.) We consider four flavors of the obfuscated program. The one in Figure 3 only offers UA; the one in Figure 4 offers identifiable abort. In both programs, if the steps **in magenta** are present, the program exploits the availability of private peer-to-peer channels in the first round; if they are omitted, the program assumes only public peer-to-peer channels.

In our constructions, each party generates a signing / verification key pair, and sends all her peers the verification key, as well as a signed PPKE encryption of her input in the first round; this encryption is under the public key  $ek$ . Informally, PPKE ensures that honest party inputs are not leaked, despite the presence of the decryption key in the obfuscated program. **If private peer-to-peer channels are available, the parties generate additional signing / verification key pairs, one for each of their peers. They send one secret signing key from such a pair privately to each peer. These help provide our obfuscated program with proofs of successful one-way communication (PoOWC).**

In the second round, all parties broadcast signed echos of all the **public** messages they received. (Each party's second round broadcast message also includes the authoritative version of her own public verification key, **and, if private peer-to-peer channels are available, all of the public verification keys she generated for the purposes of proofs of successful one-way communication.** Finally, it includes an additional ciphertext  $cttp$  (under the key  $ek'$ ); this ciphertext is only used for the proof of security, so we will not discuss it further until the proof. Her signature is produced on all of this, as well as on the function the party wishes to compute.) **If private peer-to-peer channels are available, the parties use the secret signing keys they thus received to produce additional signatures.**

The parties then use the second round broadcast messages as inputs to the obfuscated program, which gives them the output. The signed echos provide a *verifiable broadcast* of the encrypted inputs, in the sense that after the second round, each party holds a proof that sufficiently many other parties saw the same encrypted inputs. This proof is checked by the program before it produces an output, and serves to prevent the adversary from recomputing the output with different inputs. If the check passes, the program decrypts the ciphertexts and evaluates the function on the inputs thus obtained.

Figure 3.1: P2P-BC MPC with Asynchrony from Obfuscation

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

- An puncturable public key encryption scheme  $\text{PPKE} = (\text{keygen}_s, \text{keygen}_r, \text{pkeygen}_r, \text{enc}, \text{dec})$ .
  - A signature scheme  $\text{SIG} = (\text{keygen}, \text{sign}, \text{verify})$ .
- (We make liberal use of the fact that, in our construction of PPKE,  $\text{PPKE.keygen}_s$  and  $\text{SIG.keygen}$  can be the same.)

**Setup.**

- The CRS is set up as follows:
  - $(\text{ek}, \text{dk}) \leftarrow \text{PPKE.keygen}_r(1^\lambda)$ .
  - $(\text{ek}', \text{dk}') \leftarrow \text{PPKE.keygen}_r(1^\lambda)$ .
  - $\text{OP} \leftarrow \text{diO}(\text{P}_{\text{dk}, \text{dk}'})$ .
  - $(\text{ek}, \text{ek}', \text{OP})$  is published as the CRS.

**First Round.** Each party  $P_i$  does the following:

- Runs  $(\text{vk}_i, \text{sk}_i) \leftarrow \text{SIG.keygen}(1^\lambda)$ .
- **Runs  $(\text{vk}_{\text{PoOWC}(i \rightarrow j)}, \text{sk}_{\text{PoOWC}(i \rightarrow j)}) \leftarrow \text{SIG.keygen}(1^\lambda)$  for  $j \in [n]$ .**
- Encrypts her input  $x_i$  as  $c_i \leftarrow \text{PPKE.enc}(\text{sk}_i, \text{vk}_i, \text{ek}, x_i)$ .
- Signs  $c_i$ , the round number and her identity<sup>a</sup> as

$$\sigma_{i, \text{round1}} \leftarrow \text{SIG.sign}(\text{sk}_i, (c_i, \text{"round1"}, \text{"party } i\text{"})).$$

- Sends  $(\text{vk}_i, c_i, \sigma_{i, \text{round1}})$  to every other party.
- **Sends  $\text{sk}_{\text{PoOWC}(i \rightarrow j)}$  privately to party  $P_j$  for  $j \in [n]$ .**

**Second Round.** Let  $\text{vk}_{j \rightarrow i}$ ,  $c_{j \rightarrow i}$ , and  $\sigma_{j \rightarrow i, \text{round1}}$  denote the verification key, ciphertext and signature received by party  $P_i$  from party  $P_j$  over peer-to-peer channels in the first round. **Let  $\text{sk}_{\text{PoOWC}(j \rightarrow i)}$  denote the signing key received by party  $P_i$  from party  $P_j$  over private peer-to-peer channels.** Each party  $P_i$  does the following:

- Lets  $\text{pubview}_i := \{(\text{vk}_{j \rightarrow i}, c_{j \rightarrow i}, \sigma_{j \rightarrow i, \text{round1}})\}_{j \in [n]}$  denote all the public messages she received in the first round.

The next step is only necessary for the proof.

- Encrypts a null value  $\perp$  as  $\text{cttp}_i \leftarrow \text{PPKE.enc}(\text{sk}_i, \text{vk}_i, \text{ek}', \perp)$ .
- Lets  $m_i := (\text{vk}_i, \{\text{vk}_{\text{PoOWC}(i \rightarrow j)}\}_{j \in [n]}, \text{pubview}_i, \text{cttp}_i)$  denote everything she will send in this round.
- Signs  $m_i$ , as well as the function to be computed, as  $\sigma_{i, \text{round2}} \leftarrow \text{SIG.sign}(\text{sk}_i, (m_i, f, \text{"round2"}, \text{"party } i\text{"}))$ .
- **Produces additional signatures using each of the privately received signing keys as  $\sigma_{\text{PoOWC}(j \rightarrow i)} \leftarrow \text{SIG.sign}(\text{sk}_{\text{PoOWC}(j \rightarrow i)}, (m_i, f, \text{"round2"}, \text{"party } i\text{"}))$  for every  $j \in [n]$ .**

- Broadcasts  $(m_i, \sigma_{i, \text{round2}}, \{\sigma_{\text{PoOWC}(j \rightarrow i)}\}_{j \in [n]})$ .

**Output Computation.** Each party  $P_i$  computes the output as  $y \leftarrow \text{OP}(f, \{(m_j, \sigma_{j, \text{round2}}, \{\sigma_{\text{PoOWC}(k \rightarrow j)}\}_{k \in [n]})\}_{j \in [n]})$ .

---

*P2P-BC using diO.*

<sup>a</sup>We have parties sign their protocol identities to prevent key copying attacks. Whether a PKI is available or not, we assume all parties agree on an assignment of protocol identities. This is reasonable to do because in practice, communication requires e.g. a unique IP address for each participant.

```

 $P_{dk, dk'}(f, \{(m_i, \sigma_{i, \text{round}2}, \{\sigma_{\text{PoOWC}(i \rightarrow j)}\}_{j \in [n]})\}_{i \in [n]}) :$ 

for  $i \in [n]$  do
  Parse  $m_i$  as  $(vk_i, \{vk_{\text{PoOWC}(i \rightarrow j)}\}_{j \in [n]}, \text{pubview}_i, \text{cttp}_i)$ 
  Parse  $\text{pubview}_i$  as  $\{(vk_{j \rightarrow i}, c_{j \rightarrow i}, \sigma_{j \rightarrow i, \text{round}1})\}_{j \in [n]}$ 
  if  $\text{SIG.verify}(vk_i, (m_i, f, \text{"round2"}, \text{"party } i\text{"}), \sigma_{i, \text{round}2}) = \text{reject}$  then
    return abort
  end if
end for
for  $i, j \in [n]$  do
  if  $(vk_{i \rightarrow j}, c_{i \rightarrow j}, \sigma_{i \rightarrow j, \text{"round1"}}) = (\perp, \perp, \perp)$  then
     $d_{i \rightarrow j} := \text{silence}$ 
  else
     $d_{i \rightarrow j} := \text{SIG.verify}(vk_i, (c_{i \rightarrow j}, \text{"round1"}, \text{"party } i\text{"}), \sigma_{i \rightarrow j, \text{round1}})$ 
    if  $vk_{i \rightarrow j} \neq vk_i$  then
       $d_{i \rightarrow j} := \text{reject}$ 
    end if
    if  $\text{SIG.verify}(vk_{\text{PoOWC}(i \rightarrow j)}, (m_j, f, \text{"round2"}, \text{"party } j\text{"}), \sigma_{\text{PoOWC}(i \rightarrow j)}) = \text{reject}$  then
       $d_{i \rightarrow j} := \text{reject}$ 
    end if
  end if
end for
for  $i \in [n]$  do
  if  $\exists i_1, i_2$  s.t.  $c_{i \rightarrow i_1} \neq c_{i \rightarrow i_2}$  and  $d_{i \rightarrow i_1} = d_{i \rightarrow i_2} = \text{accept}$  then
    return abort
  end if
  if  $\exists j$  s.t.  $d_{i \rightarrow j} = \text{reject}$  then
    return abort
  end if
  if  $\exists M \subseteq [n]$  s.t.  $|M| > t_m$  and  $\forall j \in M, d_{i \rightarrow j} = \text{silence}$  then
    return abort
  end if
  if  $\exists D \subseteq [n]$  s.t.  $|D| > t_d$  and  $\forall j \in D, d_{j \rightarrow i} = \text{silence}$  then
    return abort
  end if
end for
for  $i \in [n]$  do
   $A_i := \{j : d_{i \rightarrow j} = \text{accept}\}_{j \in [n]}$ 
  Select any  $j \in A_i$ 
   $c_i := c_{i \rightarrow j}$ 
   $x_i \leftarrow \text{PPKE.dec}(dk, vk_i, c_i)$ 
   $\text{mtp}_i \leftarrow \text{PPKE.dec}(dk', vk_i, \text{cttp}_i)$ 
end for
if  $\exists i_1 \neq \dots \neq i_{n-t}$  s.t.  $\text{mtp}_{i_1} = \dots = \text{mtp}_{i_{n-t}} \neq \perp$  then
  return  $\text{mtp}_{i_1}$ 
else
  return  $f(x_1, \dots, x_n)$ 
end if

```

Fig. 3: Program that returns computation output (for security with UA).

*Unanimous Abort* In order to achieve unanimous abort, the program in Figure 3 performs the following checks:

1. It verifies all the signatures on the second-round messages (**including those produced using the privately communicated signing keys**), and all

```

Pdk,dk'(f, {(mi, σi,round2, {σPoOWC(j→i) }j∈[n])}i∈[n]) :

for i ∈ [n] do
  Parse mi as (vki, {vkPoOWC(i→j) }j∈[n], pubviewi, cttpi)
  Parse pubviewi as {(vkj→i, cj→i, σj→i,round1)}j∈[n]
  if SIG.verify(vki, (mi, f, "round2", "party i"), σi,round2) = reject then
    return aborti
  end if
end for
for i, j ∈ [n] do
  if (vki→j, ci→j, σi→j, "round1") = (⊥, ⊥, ⊥) then
    di→j := silence
  else
    di→j := SIG.verify(vki, (ci→j, "round1", "party i"), σi→j,round1)
    if vki→j ≠ vki then
      di→j := reject
    end if
    if SIG.verify(vkPoOWC(i→j), (mj, f, "round2", "party j"), σPoOWC(i→j)) = reject then
      di→j := reject
    end if
  end if
end for
for i ∈ [n] do
  if ∃i1, i2 s.t. ci→i1 ≠ ci→i2 and di→i1 = di→i2 = accept then
    return aborti
  end if
  Let Ri→ ⊆ [n] be the maximal set s.t. ∀j ∈ Ri→, di→j = reject.
  Let R→i ⊆ [n] be the maximal set s.t. ∀j ∈ R→i, dj→i = reject.
  Let Mi ⊆ [n] be the maximal set s.t. ∀j ∈ Mi, di→j ∈ {silence, reject}.
  Let Di ⊆ [n] be the maximal set s.t. ∀j ∈ Di, dj→i ∈ {silence, reject}.
  if |Ri→ ∪ R→i| > t then
    return aborti
  end if
  if |Mi| > tm + t then
    return aborti
  end if
  if |Di| > td + t then
    return aborti
  end if
  if |Mi ∪ Di| > td + tm + t then
    return aborti
  end if
end for
if ∃D, M ⊆ [n] s.t. D ∩ M = ∅, |D| > t, |M| > t, |D| + |M| > t + td + tm or |D| + |M| >
t + 2 min(td, tm) and ∀i ∈ D, ∀j ∈ M, dj→i ∈ {silence, reject} ∧ di→j ∈ {silence, reject}
then
  return abort
  ▷ Note that
  this will never happen, so it does not violate security with identifiable abort. There must be at
  least one honest party in each of D and M. Let hD be the honest party in D, and hM be the
  honest party in M. Let JD be the set of parties i such that dhD→i ∈ {silence, reject}; note
  that M ⊆ JD. Let JM be the set of parties i such that di→hM ∈ {silence, reject}; note that
  D ⊆ JM. In any real execution, it must be that |JD ∪ JM| ≤ t + td + tm. However, here we
  have |JD ∪ JM| ≥ |M ∪ D| > t + td + tm. The conditions in magenta will never happen
  if private peer-to-peer channels are available, by similar logic.
end if
for i ∈ [n] do
  Ai := {j : di→j = accept}j∈[n]
  Select any j ∈ Ai
  ci := ci→j
  xi ← PPKE.dec(dk, vki, ci)
  mtpi ← PPKE.dec(dk', vki, cttpi)
end for
if ∃i1 ≠ ... ≠ in-t s.t. mtpi1 = ... = mtpin-t ≠ ⊥ then
  return mtpi1
else
  return f(x1, ..., xn)
end if

```

Fig. 4: Program that returns computation output (for security with IA).

echoed signatures on the first-round messages, under the keys provided in the second-round messages.

2. It checks that there do not exist verifying signatures on two different ciphertexts from the same party.
3. It checks that at least  $n - t_m$  parties echoed each party's first-round message, and that each party echoed at least  $n - t_d$  other parties' first-round messages. **(If private peer-to-peer channels are available, a party  $P_i$  is considered to have successfully echoed party  $P_j$ 's first-round message *only if* she signs her second-round message with the signing key she received privately from  $P_j$ .)**

**Checking the successful communication via first-round private channels helps obtain better bounds, as shown in Theorem 14, because not seeing all the messages a given honest party receives helps make it difficult for the adversary to produce fake second-round messages on behalf of that party.**

**Theorem 13 (PubP2P-BC, UA, CRS,  $t + t_d + t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and let  $t + t_d + t_m < n$ . Then, the protocol described in Figure 3.1 with the program described in Figure 3 (excluding the steps **in magenta**) is a PubP2P-BC, UA, CRS protocol that securely computes  $f$  over two rounds, the first of which is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over synchronous broadcast and peer-to-peer channels.*

**Theorem 14 (PrivP2P-BC, UA, CRS,  $t + 2 \min(t_d, t_m) < n, t + t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and let  $t + 2 \min(t_d, t_m) < n, t + t_m < n$ . Then, the protocol described in Figure 3.1 with the program described in Figure 3 (including the steps **in magenta**) is a PrivP2P-BC, UA, CRS protocol that securely computes  $f$  over two rounds, the first of which is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over synchronous broadcast and peer-to-peer channels.*

We now argue that, after an execution of the protocol in Figure 3.1, the adversary can only run the obfuscated program in Figure 3 on one set of inputs any of which belong to honest parties without getting an abort. We argue this in two parts: first, why the adversary is unable to use only some of the honest parties' inputs, and second, why the adversary is unable to substitute corrupt parties' inputs.

### **Honest Party Second Round Messages Must All Be Used Together**

The adversary cannot drop honest party second-round messages and still obtain the output; if she tries this, she will simply get an abort as a result of the first if-statement. The adversary might hope to be able to replace some honest parties' second-round messages with different second-round messages,

along with a different verification keys. (Note that it's very important that the adversary be unable to take an honest party ciphertext and use it together with a corrupt party verification key in a way where information about the honest party's input is preserved. Obs 3 shows that PPKE guarantees that swapping in a different verification key causes a ciphertext to decrypt to something unrelated to the original message.) Such an attack by the adversary results in a partitioning of honest parties into two groups: the group  $M$  of parties whose second-round messages remain untouched, and the group  $D$  of parties whose second-round messages got replaced. In order not to get rejected by the second if-statement of the third for-loop, the adversary must also prevent first-round communication from  $D$  to  $M$  (**and either also prevent first-round communication from  $M$  to  $D$ , or claim, in the new second-round messages of  $D$ , that this communication was prevented**). For each  $i \in D$  and  $j \in M$ , we then have  $\mathbf{d}_{j \rightarrow i} = \mathbf{d}_{i \rightarrow j} = \text{silence}$ . If  $|D| > t_m$  **or**  $|M| > t_m$ , then the third if-statement of the third for-loop will abort; if  $|M| > t_d$  **or**  $|D| > t_d$ , then the fourth if-statement of the third for-loop will abort. If  $t_d + t_m < n - t$ ,  $2t_d < n - t$  **or**  $2t_m < n - t$ , then we must have either  $|D| > t_m$ ,  $|D| > t_d$ ,  $|M| > t_d$ , or  $|M| > t_m$ , so an abort will happen.

**$\mathcal{A}$  is Unable to Substitute Corrupt Party Inputs While Keeping Honest Party Second Round Messages** If different signatures on different ciphertexts verify under the same key, then the first if-statement of the third for-loop will abort. If any honest parties receive a non-verifying signature, then the second if-statement of the third for-loop will abort. If more than  $t_m$  parties echo  $\perp$ , then the third if-statement of the third for-loop will abort. So, if an abort does not happen, then at most  $t_m$  honest parties can echo  $\perp$  from a corrupt party, and the rest echo one fixed ciphertext. As long as  $n - t - t_m > 0$  (which is the number of honest parties minus  $t_m$ ), this guarantees that at least one honest party echoed each corrupt party ciphertext. So, the adversary cannot swap out a corrupt party ciphertext for a different one (without swapping out at least one honest party second-round message, which requires swapping out *all* honest party second-round messages, as per the above argument); if the adversary attempts this, the first if-statement of the third for-loop will abort.

We conclude that the adversary can only evaluate the program on one set of corrupt party ciphertexts with the honest parties' set of ciphertexts.

*Proof (of Theorems 13 **and** 14).* We prove security by building a simulator. Recall that we allow each party  $P_i$  to include a ciphertext  $\text{cttp}$ , encrypted to the obfuscated program's key  $\mathbf{ek}'$ , within with their signed and broadcast second-round message. In a real execution, honest parties will encrypt  $\perp$ , and these ciphertexts will be ignored by the program. However, we will use these ciphertexts to allow simulated honest parties to communicate the output of the function to the program.

We now proceed to provide a proof sketch of the indistinguishability between the ideal and the real execution through a sequence of hybrids.

1. We start with a real execution; the simulator honestly plays the role of all honest parties, given their inputs. Notice that the simulator runs the setup phase, including the generation of the program's keys. As a consequence, the simulator knows the secret key  $\mathbf{dk}$  and is able to use it to extract the corrupt parties' inputs from their ciphertexts; details follow.

The simulator does all the checks in the same way as the program  $\text{OP}$  described in Figure 3 does. If these checks go through, each corrupted party sent the same ciphertext to at least  $n - (t + t_m)$  other honest parties. The simulator uses this ciphertext to extract the input of the corrupted party.

2. During the CRS setup, the simulator punctures the key  $(\mathbf{ek}', \mathbf{dk}')$  (used to decrypt  $\text{cttp}$ ) at the set of honest party verification keys  $\{\mathbf{vk}_i\}_{i \in \mathbb{H}}$ .  $\mathbf{ek}'$  is indistinguishable from that in the previous hybrid by the puncture hiding property of PPKE. By the security of  $\text{diO}$ , this hybrid is indistinguishable from the previous one, since the program output does not change for any inputs that are computationally feasible for the adversary to find (as  $\text{cttp}$  does not affect the output in this hybrid).
3. The simulator, having extracted the corrupt parties' inputs, sends them to the trusted party and obtains the function output  $y$ .

The simulated honest parties set  $\text{cttp}_i \leftarrow \text{PPKE.enc}(\mathbf{ek}', \mathbf{vk}_i, \mathbf{sk}_i, y)$ . By the secure puncturability of PPKE, this hybrid is indistinguishable from the previous one, since the decryption key  $\mathbf{dk}'$  is punctured at the honest party verification keys, and thus all their ciphertexts always decrypt to  $\perp$ .

The simulator computes all the steps of  $\text{P}_{\mathbf{dk}, \mathbf{dk}'}$ . If it gets  $y$  as output, it sends `continue` to the trusted party; otherwise, it forwards the returned abort message.

4. The simulator goes back to not puncturing  $\mathbf{dk}'$ . Now, the obfuscated program will be using the messages  $\text{mtp}_i$  to determine the output. By our earlier argument that honest party second-round messages must all be used together and that corrupt party inputs cannot be substituted, the adversary cannot find any inputs on which the program in this hybrid and the program in the previous hybrid differ, and so is limited to only one set of inputs (modulo signature randomness) that gets him past the checks. That set of inputs results in the same output from both programs. It follows that program in the previous hybrid will only ever evaluate  $f$  on those inputs; the program in this hybrid will return the output of  $f$  on those inputs as instructed by the (simulated) honest parties (who are at least  $n - t$ ) through  $\text{mtp}$ . So, by the security of  $\text{diO}$ , the two obfuscations are indistinguishable.
5. All that remains is for us to get rid of encryptions of the honest parties' inputs. The simulator accomplishes this by puncturing  $\mathbf{dk}$  at the set of honest party verification keys  $\{\mathbf{vk}_i\}_{i \in \mathbb{H}}$ . The new program is indistinguishable from the previous one by the puncture hiding property of PPKE and by the security of  $\text{diO}$ .
6. Finally, we have the honest parties encrypt  $\perp$  instead of their inputs. By the secure puncturability of PPKE, this hybrid is indistinguishable from the previous one.

Note that this final hybrid no longer requires the simulator to know the inputs of the honest parties.

*Identifiable Abort* In order to achieve identifiable abort, the program in Figure 4 performs the following checks:

1. It verifies all the signatures on the second-round messages produced using the parties' own keys. If a signature does not verify, it blames the owner of the bad signature.
2. It checks that there do not exist verifying signatures on two different ciphertexts from the same party. If this happens, it blames that party.
3. For each party  $P_i$ , it checks that for at most  $t$  other parties  $P_j$ , either  $P_j$  echoed a non-verifying signature from  $P_i$ , or  $P_i$  echoed a non-verifying signature from  $P_j$ . Otherwise, it blames  $P_i$ .
4. For each party  $P_i$ , it checks that at least  $n - t - t_m$  parties successfully echoed  $P_i$ 's correct ciphertext, and  $P_i$  successfully echoed at least  $n - t - t_d$  others' correct ciphertexts. It also checks that both of these things happened for at least  $n - t - t_d - t_m$  others. Otherwise, it blames  $P_i$ . **(As before, if private peer-to-peer channels are available, a party  $P_i$  is considered to have successfully echoed party  $P_j$ 's ciphertext *only if* she signs her second-round message with the signing key she received privately from  $P_j$ .)**
5. Finally, the program aborts *blaming no-one* if in one special case that can never happen in a real execution, and thus it is not a problem to abort there. This case is described in detail in Figure 4.

An important caveat is that the last check takes time exponential in the number of parties. This limits our constructions with identifiable abort to support only a constant number of parties.

**Theorem 15 (PubP2P-BC, IA, CRS,  $3t + \max(t_d, t_m) < n, 2t + t_d + t_m < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function for constant  $n$ , and let  $3t + \max(t_d, t_m) < n$ . Then, the protocol described in Figure 3.1 with the program described in Figure 4 (excluding the steps **in magenta**) is a PubP2P-BC, UA, CRS protocol that securely computes  $f$  over two rounds, the first of which is over public peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over synchronous broadcast and peer-to-peer channels.*

**Theorem 16 (PrivP2P-BC, IA, CRS,  $3t + t_m < n, 2t + 2 \min(t_d, t_m) < n$ ).** *Let  $f$  be an efficiently computable  $n$ -party function for constant  $n$ , and let  $3t + t_m < n, t_m \leq t$ . Then, the protocol described in Figure 3.1 with the program described in Figure 4 (including the steps **in magenta**) is a PrivP2P-BC, IA, CRS protocol that securely computes  $f$  over two rounds, the first of which is over private peer-to-peer channels with  $(t_d, t_m)$ -asynchrony, and the second of which is over synchronous broadcast and peer-to-peer channels.*

As before, we argue that, after an execution of the protocol in Figure 3.1, the adversary can only run the obfuscated program in Figure 4 on one set of inputs any of which belong to honest parties without getting an abort.

### Honest Party Second Round Messages Must All Be Used Together

The adversary cannot drop honest party second-round messages and still obtain the output; if she tries this, she will simply get an abort as a result of the first if-statement. The adversary might hope to be able to replace some honest parties' second-round messages with different second-round messages, along with different verification keys. Such an attack by the adversary results in a partitioning of honest parties into two groups: the group  $M$  of parties whose second-round messages remain untouched, and the group  $D$  of parties whose second-round messages got replaced. In order not to get rejected by the second if-statement, the adversary must prevent first-round communication from all but  $t$  members of  $D$  to each member of  $M$ , and to all but  $t$  members of  $M$  from each member of  $D$ . This is true since every message that is successfully delivered from a member of  $D$  to a member of  $M$  triggers a rejection once the second round messages of  $D$  are replaced.

If  $|D| > t + t_m$  **or**  $|M| > t + t_m$ , then the third if-statement of the third for-loop will abort; if  $|M| > t + t_d$  **or**  $|D| > t + t_d$ , then the fourth if-statement of the third for-loop will abort.

If  $|M| > t$ ,  $|D| > t$  and  $|M| + |D| > t + t_d + t_m$  **or**  $|M| + |D| > t + 2 \min(t_d, t_m)$ , then the program will abort (blaming no-one, since such a situation cannot happen in the real world). In order to make sure that if the third and fourth if-statements go untriggered, this abort *is* triggered, we need that  $n - t > 2t + \max(t_d, t_m)$  (**or, if private channels are available,  $n - t > 2t + \min(t_d, t_m)$** ) and  $n - t > t + t_d + t_m$ .

(We explain the last observation in the case where private peer-to-peer channels are not available. If the third and fourth if-statements go untriggered, we know that  $|D| \leq t + t_d$  and  $|M| \leq t + t_m$ . In order to force  $|D|$  to be larger than  $t$ , we need  $|D| + |M| = n - t > t + t + t_d = 2t + t_d \Rightarrow 3t + t_d < n$ . Symmetrically, in order to force  $|M|$  to be larger than  $t$ , we need  $|D| + |M| = n - t > t + t + t_m = 2t + t_m \Rightarrow 3t + t_m < n$ . We conclude that we need  $3t + \max(t_d, t_m) < n$ . We also need  $|D| + |M| = n - t \geq t + t_d + t_m \Rightarrow 2t + t_d + t_m < n$ .

**We now explain the last observation in the case where private peer-to-peer channels are available. If the third and fourth if-statements go untriggered, we know that  $|D|, |M| \leq t + \min(t_d, t_m)$ ; in order to force both  $|D|$  and  $|M|$  to be larger than  $t$ , we need  $|D| + |M| = n - t > t + \min(t_d, t_m) + t = 2t + \min(t_d, t_m) \Rightarrow 3t + \min(t_d, t_m) < n$ . We also need  $|D| + |M| = n - t \geq t + 2 \min(t_d, t_m)$ .**

### $\mathcal{A}$ is Unable to Substitute Corrupt Party Inputs While Keeping Honest Party Second Round Messages

Consider corrupt party  $a$ . If different signatures on different ciphertexts verify under party  $a$ 's key, then the first if-statement will abort; if  $> t$  honest parties reject party  $a$ , then the second if-statement will abort. Let  $A$  be the set of parties who accept party  $a$ . If the adversary substitutes that corrupt party's input, then all honest parties

who accepted before will reject now. As long as  $|A| > t$ , this is ok, and does not allow input substitution (thanks to the second if-statement). As long as  $n - t > 2t + t_m$ , such input substitution is impossible.

The description of the simulator and the proof of indistinguishability between real and ideal execution are very similar to the one of Theorem 13, with modifications to the corresponding thresholds.

*Remark 1.* We would, of course, prefer to use indistinguishability obfuscation rather than differing-inputs obfuscation, since indistinguishability obfuscation is a weaker (more realistic) assumption. Informally, while differing-inputs obfuscation guarantees that the obfuscations of two programs are indistinguishable if it's hard to find inputs on which they differ, indistinguishability obfuscation only guarantees that the obfuscations of two programs are indistinguishable if *there do not exist* inputs on which they differ. One might think that the use of signatures in our program is an insurmountable obstacle to basing security on indistinguishability obfuscation, since there always *exist* signatures on any message — they are just hard to find. However, Boneh and Zhandry [BZ14] introduce *constrained* signatures, where it is possible to limit the domain of messages on which signatures exist (in a way where the public verification key does not reveal this domain). Unfortunately, we cannot leverage constrained signatures in our proof, since the public verification keys (which determine the domain of messages on which signatures exist) are fixed when the program is published during setup, while the desired message domain is determined by corrupt parties' messages in the first round, since honest parties are obliged to sign echos of corrupt party messages.

## 4 BC-P2P

In this section, we explore the setting in which the first round communication (over the broadcast and peer-to-peer channels) is assumed to be synchronous and the second round communication (over peer-to-peer channels) is assumed to be asynchronous. Here, we only consider asynchrony in the form of *arbitrary message scheduling* since the issue of input deprivation does not occur in BC-P2P protocols, because the first round communication is synchronous (therefore, an honest party's input can always be included in output computation since her first round messages will be received by everyone). We summarize our results in Table 1.

### 4.1 Lower Bounds

In this section, we show two negative results. First, we show that even the weakest form of security — security with selective abort — is impossible in the BC-P2P setting when  $n \leq 2t$ , where the first round communication is synchronous (over broadcast and private peer-to-peer channels) and the second round peer-to-peer communication is asynchronous. This holds even if parties have access to a PKI.

To prove the impossibility, we show that any such protocol must be such that a set of  $t$  parties would learn the output at the end of first round itself, which makes the protocol vulnerable to a residual attack.

Next, we show that selective abort remains impossible for arbitrary  $n$  and  $t \geq 1$  in the PubBC-P2P setting when the first round communication is synchronous over broadcast and *public* peer-to-peer channels and the second round peer-to-peer communication is asynchronous. At a high-level, we argue that any such protocol must be such that the output can be computed without using  $t$  of the second-round messages (in an execution where everyone behaves honestly), which contradicts an impossibility result of Damgård *et al.* [DRSY23].

### With PKI

**Theorem 17 (BC-P2P, SA, PKI,  $n \leq 2t$ ).** *There exist functions  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with selective abort if  $n \leq 2t$ ; where the first round communication is synchronous over broadcast and peer-to-peer channels and the second round communication is asynchronous over peer-to-peer channels.*

*Proof.* Towards a contradiction, assume protocol  $\Pi$  that achieves selective abort against  $t \geq \frac{n}{2}$  corruptions, where the first round communication is synchronous (over broadcast and peer-to-peer channels) and the second round peer-to-peer communication is asynchronous. Consider an execution of  $\Pi$  where everyone behaves honestly. Due to correctness, this must result in all parties obtaining the correct output with overwhelming probability. Since the second round communication is asynchronous, an honest party, say  $P_i$ , computes her output after receiving messages from  $n - t$  parties (say, constituting the set  $S$ , where  $P_i \in S$ ) in Round 2. This means that the combined view of parties in  $S$  at the end of Round 1 itself was sufficient to compute the protocol output with overwhelming probability. This makes the protocol vulnerable to the following residual attack: Consider a scenario where the adversary passively corrupts the parties in  $S$ , where  $|S| = (n - t) \leq t$ . The parties in  $S$  can compute the output at the end of Round 1. Therefore, the passive adversary can plug in various inputs of its choice on behalf of the parties in  $S$  and obtain multiple evaluations of the function, while the inputs of the honest parties remain fixed. This breaches privacy of honest parties, contradicting the security of  $\Pi$ . We refer to Theorem 1 for concrete example of  $f$  (namely  $f_{\text{mot}}$ ), where we demonstrate how multiple evaluations of a function violates privacy.

**Without PKI, Without Private Channels** Before presenting the proof, we recap a useful definition and theorem from [DMR<sup>+</sup>21,DRSY23].

**Definition 5 (Last Message Resiliency [DMR<sup>+</sup>21]).** *A protocol is  $r$ -last message resilient if, in an honest execution, any protocol participant  $P_i$  can compute its output without using  $r$  of the messages it received in the last round.*

**Theorem 18.** [DRSY23] *Assume parties have access to a CRS, but a PKI is unavailable. There exists a function  $f$  such that any two-round protocol  $\Pi$  securely realizing  $f$  whose first round can be executed over broadcast and public peer-to-peer channels cannot be  $r$ -last message resilient for  $r > 0$ .*

**Theorem 19 (PubBC-P2P, SA, CRS,  $t \geq 1$ ).** *There exists a function  $f$  such that no  $n$ -party two-round protocol can compute  $f$  with selective abort against  $t \geq 1$  corruptions, where the first round communication is synchronous (over broadcast and public peer-to-peer channels) and the second round (private) peer-to-peer communication is asynchronous.*

*Proof.* Towards a contradiction, assume protocol  $\Pi$  that achieves selective abort against  $t \geq 1$  corruptions, where the first round communication is synchronous (over broadcast and public peer-to-peer channels) and the second round peer-to-peer communication is asynchronous. Correctness demands that  $\Pi$  must be such that when everyone behaves honestly, a party is able to compute the output (with overwhelming probability) upon receiving messages from just  $(n - t)$  parties in the second round. In other words,  $\Pi$  must be  $t$ -last message resilient (Definition 5). However, this contradicts the impossibility of Damgård *et al.* [DRSY23] which proves that a protocol whose first round communication is over broadcast and public peer-to-peer channels cannot be  $r$ -last message resilient for  $r > 0$  (Theorem 18). We have thus arrived at a contradiction, completing the proof of Theorem 19.

## 4.2 Upper Bounds

In this section, we first formally state the observation in [RU21] that when a PKI is available and  $n > 2t$ , even the strongest guarantee of GOD can be achieved. More specifically, Rambaud and Urban [RU21] make the simple observation that the BC-P2P, GOD, PKI,  $n > 2t$  protocol of [DMR<sup>+</sup>21] also extends to the setting when the second round peer-to-peer communication is asynchronous.

Next, we show that when a PKI is not available (only a CRS and private peer-to-peer channels are available) and  $n > 2t$ , selective abort can be achieved. This is the best one can hope for as the works of [PR18,DRSY23] show that PrivBC-P2P UA CRS is impossible even in the fully synchronous setting. Further, such a construction must use private peer-to-peer channels in the first round, following the impossibility result of Theorem 19 (which shows that even selective abort cannot be achieved if the first round peer-to-peer communication is public).

### With PKI

**Observation 4 (BC-P2P, GOD, PKI,  $n > 2t$  [RU21])** *Let  $f$  be an efficiently computable  $n$ -party function and let  $n > 2t$ . Suppose  $\Pi_{bc}$  is a two broadcast-round protocol that securely computes the function  $f$  with guaranteed output delivery with the additional constraint that the straight-line simulator can extract inputs from the first-round messages and it is efficient to check whether*

a given second-round message is correct. Then there exists a BC-P2P, GOD, PKI protocol that securely computes  $f$  over two rounds, the first of which is over synchronous broadcast and peer-to-peer channels and the second of which is over asynchronous peer-to-peer channels.

### Without PKI, With Private Channels

**Theorem 20 (PrivBC-P2P, SA, CRS,  $n > 2t$ ).** *Let  $f$  be an efficiently computable  $n$ -party function and let  $n > 2t$ . Suppose  $\Pi_{\text{sl-abort}}$  is a PrivBC-P2P protocol that securely computes the function  $f$  with selective abort satisfying the following additional constraints – (1) there exists a straight-line simulator that can extract inputs from the first-round messages (2) it is efficient to check whether a given second-round message is correct and (3)  $(n - t)$  correct second-round messages are sufficient to compute the output in an execution where everyone behaves honestly. Then there exists protocol  $\Pi_{\text{sl-abort}}^{\text{async}}$  that achieves the same guarantee of selective abort when the second round peer-to-peer communication is asynchronous.*

*Proof (Sketch).* Starting from  $\Pi_{\text{sl-abort}}$ , it is possible to define  $\Pi_{\text{sl-abort}}^{\text{async}}$  with the following modifications: (a) the second round peer-to-peer communication is sent over asynchronous peer-to-peer channels and (b) In the second round, honest parties efficiently check if the second round message they receive is generated correctly (possible due to property (2)), wait for  $(n - t)$  correctly generated messages and compute the output based on them.

Correctness holds when everyone behaves honestly, as property (3) ensures that the honest parties are able to compute the output. The simulator for  $\Pi_{\text{sl-abort}}^{\text{async}}$  can proceed identical to the simulator for  $\Pi_{\text{sl-abort}}$ , since the simulator extracts the input from the first (broadcast) round and it is unaffected by the asynchrony in the second round.

We argue now that the the two-round construction of Ananth *et al.* [ACGJ18], say  $\Pi_{\text{god}}^{\text{sm}}$ , that achieves guaranteed output delivery against  $t < n/2$  semi-malicious fail-stop corruptions can be modified to obtain  $\Pi_{\text{sl-abort}}$ . Intuitively, this protocol is a promising starting point because its GOD guarantee is useful to achieve property (3) i.e., we require that  $(n - t)$  second-round messages are sufficient to compute the output.

At a high-level, Ananth *et al.* present a compiler that compiles a two-round broadcast semi-malicious protocol  $\phi$  into  $\Pi_{\text{god}}^{\text{sm}}$ . The compiler proceeds as follows: In the first round of  $\Pi_{\text{god}}^{\text{sm}}$ , each party broadcasts the first round of  $\phi$ . Additionally, she garbles her second-message function from  $\phi$ , which has her own input hardcoded, and takes as input all the first-round messages she receives. Further, she secret-shares the input labels for her garbled circuit with threshold  $t < n/2$  and sends to other parties their respective shares using the private peer-to-peer channels in the first round. In the second round of  $\Pi_{\text{god}}^{\text{sm}}$ , parties simply broadcast the appropriate shares of the garbled circuit, based on the first-round messages received via broadcast. This protocol achieves guaranteed output delivery against  $t < n/2$  semi-malicious fail-stop corruptions, because even if  $t$  parties abort in

the second round, the shares obtained from the  $(n - t) > t$  remaining parties is sufficient to reconstruct the appropriate labels.

Next, we describe the modifications to the compiler required to upgrade  $\Pi_{\text{god}}^{\text{sm}}$  to malicious security with selective abort (i.e.  $\Pi_{\text{sl-abort}}$ ) – A maliciously secure two-round broadcast protocol with abort security (such as the protocols of [BL18][GS18]) is used as the underlying two-round broadcast protocol to be compiled. In the first round, we make the parties broadcast a set of commitments corresponding to the secret shares and send the openings of the relevant commitment to the recipient of the corresponding share over private peer-to-peer channels. Further, the parties are required to broadcast a NIZK proof to ensure correctness of their actions in the first round. In the second round, a party sends an “abort” message to all the other parties and outputs  $\perp$  if any of the NIZK proofs failed or she received an invalid opening. Otherwise, she sends the relevant shares (similar to  $\Pi_{\text{god}}^{\text{sm}}$ ), with an accompanying NIZK proof showing correctness of her second-round message over peer-to-peer channels. A party proceeds to reconstruction of the labels and evaluating the next-message garbled circuits only if all the NIZK proofs she obtained verify and she did not receive “abort” message from any party.

We claim that  $\Pi_{\text{sl-abort}}$  satisfies the properties outlined in Theorem 20. Properties (1) and (2) are satisfied as 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. Lastly, property (3) is satisfied as when everyone behaves honestly, the second round messages of  $(n - t) > t$  parties is sufficient to proceed to output computation. This completes the overview of the construction of  $\Pi_{\text{sl-abort}}$ .

Security of  $\Pi_{\text{sl-abort}}$  can be shown by augmenting the security proof of  $\Pi_{\text{god}}^{\text{sm}}$  with additional arguments that rely on the security of commitments and NIZKs. Lastly, we point that even though the second-round messages of  $\Pi_{\text{sl-abort}}$  are sent over peer-to-peer channels (unlike  $\Pi_{\text{god}}^{\text{sm}}$ ), this cannot lead to parties obtaining an incorrect output or a pair of honest parties obtaining different non- $\perp$  outputs since the input is extracted from the first round itself. We refer to Appendix D for a formal description of  $\Pi_{\text{sl-abort}}$  and further details.

## References

- ABG<sup>+</sup>13. Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. <http://eprint.iacr.org/2013/689>.
- ACGJ18. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 395–424. Springer, Heidelberg, August 2018.
- ACGJ19. Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Two round information-theoretic MPC with malicious security. In

- Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 532–561. Springer, Heidelberg, May 2019.
- BCG93. Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th Annual ACM Symposium on Theory of Computing*, pages 52–61. ACM Press, May 1993.
- BGI<sup>+</sup>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 Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, Heidelberg, August 2001.
- BHR12a. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 134–153. Springer, Heidelberg, December 2012.
- BHR12b. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 784–796. ACM Press, October 2012.
- BKR94. Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM Symposium Annual on Principles of Distributed Computing*, pages 183–192. Association for Computing Machinery, August 1994.
- BL18. Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 500–532. Springer, Heidelberg, April / May 2018.
- BSW16. Mihir Bellare, Igors Stepanovs, and Brent Waters. New negative results on differing-inputs obfuscation. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 792–821. Springer, Heidelberg, May 2016.
- BZ14. Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 480–499. Springer, Heidelberg, August 2014.
- CGZ20. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 828–858. Springer, Heidelberg, May 2020.
- Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th Annual ACM Symposium on Theory of Computing*, pages 364–369. ACM Press, May 1986.
- DDO<sup>+</sup>01. Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of

- Lecture Notes in Computer Science*, pages 566–598. Springer, Heidelberg, August 2001.
- DMR<sup>+</sup>21. Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Broadcast-optimal two round MPC with an honest majority. In *Crypto*, Lecture Notes in Computer Science, pages 155–184. Springer, Heidelberg, 2021.
- DRSY23. Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Minimizing setup in broadcast-optimal two round MPC. In *EUROCRYPT*, 2023.
- ELG84. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO’84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, Heidelberg, August 1984.
- GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 63–82. Springer, Heidelberg, August 2015.
- GM17. Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 581–612. Springer, Heidelberg, August 2017.
- GS18. Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 468–499. Springer, Heidelberg, April / May 2018.
- HNP05. Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer, Heidelberg, May 2005.
- HNP08. Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 473–485. Springer, Heidelberg, July 2008.
- IKKP15. Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 359–378. Springer, Heidelberg, August 2015.
- IKP10. Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 577–594. Springer, Heidelberg, August 2010.

- Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO’91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, Heidelberg, August 1992.
- PR18. Arpita Patra and Divya Ravi. On the exact round complexity of secure three-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 425–458. Springer, Heidelberg, August 2018.
- RU21. Matthieu Rambaud and Antoine Urban. Almost-asynchronous mpc under honest majority, revisited. Cryptology ePrint Archive, Paper 2021/503, 2021. <https://eprint.iacr.org/2021/503>.
- Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.
- Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society Press, November 1982.
- ZHM09. Vassilis Zikas, Sarah Hauser, and Ueli M. Maurer. Realistic failures in secure multi-party computation. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 274–293. Springer, Heidelberg, March 2009.

## A Preliminaries

### A.1 Differing-Inputs Obfuscation

A differing-inputs obfuscation scheme, introduced by Barak *et al.* [BGI<sup>+</sup>01], enables a party to “hide” a program in such a way that it can still be evaluated. Indistinguishability obfuscation (iO) demands that the obfuscation of two programs with identical behaviors be indistinguishable; in contrast, differing-inputs obfuscation (diO) demands that the obfuscation of two programs *for which it is hard to find an input on which the outputs differ* be indistinguishable.

Below, we restate the definitions of differing-input obfuscation due to Ananth *et al.* [ABG<sup>+</sup>13]. Before we state the definition of diO, we first define a *differing-inputs circuit family*, which is a family of circuit pairs such that it is hard to find an input on which two circuits in a randomly sampled pair differ.

**Definition 6 (Differing-Inputs Circuit Family).** A differing-inputs circuit family is a circuit family  $\mathcal{C}$  associated with an efficient algorithm  $\text{Sampler}$  such that for every efficient adversary  $\mathcal{A}$ , there exists a negligible function  $\text{negl}$  in the security parameter  $\lambda$  such that

$$\Pr[\mathcal{C}_0(x) \neq \mathcal{C}_1(x) : (\mathcal{C}_0, \mathcal{C}_1, \text{aux}) \leftarrow \text{Sampler}(1^\lambda), x \leftarrow \mathcal{A}(1^\lambda, \mathcal{C}_0, \mathcal{C}_1, \text{aux})] \leq \text{negl}(\lambda).$$

**Definition 7 (Differing-Inputs Obfuscation (diO)).** A differing-inputs obfuscation (diO) scheme is an efficient algorithm  $\text{diO}$  that takes in a circuit or program  $\mathcal{C}$ , and outputs an obfuscated circuit or program  $\text{OC}$ .

We require the following properties of diO:

*Correctness.* We say diO satisfies correctness if for all circuits  $\mathcal{C} \in \mathcal{C}$ , for all inputs  $x$ , we have that

$$\Pr[\text{OC}(x) = \mathcal{C}(x) : \text{OC} \leftarrow \text{diO}(\mathcal{C})] = 1.$$

*Polynomial Slowdown.* We say that diO satisfies polynomial slowdown if for any circuit or program  $\mathcal{C}$ , the size and running time of  $\text{diO}(\mathcal{C})$  is only polynomially larger than that of  $\mathcal{C}$ .

*Differing-Inputs.* We say that diO satisfies differing-inputs if for any (not necessarily uniform) PPT adversary  $\mathcal{A}$ , for  $(\mathcal{C}_0, \mathcal{C}_1, \text{aux}) \leftarrow \text{Sampler}(1^\lambda)$ , we have that

$$|\Pr[\mathcal{A}(\text{diO}(\mathcal{C}_0), \text{aux}) = 1] - \Pr[\mathcal{A}(\text{diO}(\mathcal{C}_1), \text{aux}) = 1]| \leq \text{negl}(\lambda).$$

### A.2 Threshold Secret Sharing Scheme

A  $t$ -out-of- $n$  secret sharing scheme allows a party to “split” a secret into  $n$  shares that can be distributed among different parties. To reconstruct the original secret  $x$  at least  $t + 1$  shares need to be used.

**Definition 8 (Secret Sharing).** A  $t$ -out-of- $n$  secret sharing scheme is a tuple of efficient algorithms  $(\mathbf{share}, \mathbf{reconstruct})$  defined as follows.

$\mathbf{share}(x) \rightarrow (s_1, \dots, s_n)$ : The randomized algorithm  $\mathbf{share}$  takes as input a secret  $x$  and output a set of  $n$  shares.

$\mathbf{reconstruct}(\{s_i\}_{i \in S \subseteq [n], |S| > t}) \rightarrow x$ : The reconstruct algorithm  $\mathbf{reconstruct}$  takes as input a vector of at least  $t + 1$  shares and outputs the secret  $x$ .

We require the following properties of a  $t$ -out-of- $n$  secret sharing scheme:

*Perfect Correctness.* The perfect correctness property requires that the shares of a secret  $x$  should always reconstruct to  $x$ . More formally, a secret sharing scheme is perfectly correct if for any secret  $x$ , for any subset  $S \subseteq [n], |S| > t$ ,

$$\Pr \left[ x = x' : \begin{array}{l} (s_1, \dots, s_n) \leftarrow \mathbf{share}(x) \\ x' \leftarrow \mathbf{reconstruct}(\{s_i\}_{i \in S}) \end{array} \right] = 1,$$

where the probability is taken over the random coins of  $\mathbf{share}$ . Moreover, if a negligible error probability is allowed, we simply say that the scheme is correct.

*Privacy.* The privacy property requires that any combination of up to  $t$  shares should leak no information about the secret  $x$ . More formally, we say that a secret sharing scheme is private if for all (unbounded) adversaries  $\mathcal{A}$ , for any set  $\mathbb{A} \subseteq \{1, \dots, n\}, |\mathbb{A}| \leq t$  and any two secrets  $x_0, x_1$  (such that  $|x_0| = |x_1|$ ),

$$\Pr \left[ \mathcal{A}(\mathbf{s}) = 1 : \begin{array}{l} \{s_i\}_{i \in [n]} = \mathbf{share}(x_0); \\ \mathbf{s} = \{s_i\}_{i \in \mathbb{A}} \end{array} \right] \equiv \Pr \left[ \mathcal{A}(\mathbf{s}) = 1 : \begin{array}{l} \{s_i\}_{i \in [n]} = \mathbf{share}(x_1); \\ \mathbf{s} = \{s_i\}_{i \in \mathbb{A}} \end{array} \right].$$

*Share Simulatability.* Additionally, we require an efficient simulator for the generated shares. More formally, we say that a secret sharing scheme is share simulatable if there exists a PPT simulator  $\mathbf{simshare}$  such that for every PPT adversary  $\mathcal{A}$ , for any set  $\mathbb{A} \subseteq \{1, \dots, n\}, |\mathbb{A}| \leq t$  (and  $\mathbb{H} = \{1, \dots, n\} \setminus \mathbb{A}$ ), and any two secrets  $x_0, x_1$ , for  $(s_0, \dots, s_n) \leftarrow \mathbf{share}(x_0), (s'_1, \dots, s'_n) \leftarrow \mathbf{share}(x_1)$  and  $\{s''_i\}_{i \in \mathbb{H}} \leftarrow \mathbf{simshare}(\{s_i\}_{i \in \mathbb{A}}, x_0)$ ,

$$|\Pr [\mathcal{A}(\{s_i\}_{i \in \mathbb{A}}, \{s_i\}_{i \in \mathbb{H}}) = 1] - \Pr [\mathcal{A}(\{s_i\}_{i \in \mathbb{A}}, \{s''_i\}_{i \in \mathbb{H}}) = 1]| \leq \text{negl}(\lambda).$$

*Instantiation.* In our constructions, we use the Shamir’s threshold secret sharing scheme [Sha79], and refer to its algorithms as  $(\mathbf{Shamir.s}, \mathbf{Shamir.reconstruct})$ .

### A.3 Garbling Scheme

A garbling scheme, introduced by Yao [Yao82] and formalized by Bellare *et al.* [BHR12b], enables a party to “encrypt” or “garble” a circuit in such a way that it can be evaluated on inputs — given tokens or “labels” corresponding to those inputs — without revealing what the inputs are.

**Definition 9 (Garbling Scheme).** A projective garbling scheme is a tuple of efficient algorithms  $\text{GC} = (\text{garble}, \text{eval})$  defined as follows.

$\text{garble}(1^\lambda, \mathbf{C}) \rightarrow (\text{GC}, \mathbf{K})$ : The garbling algorithm  $\text{garble}$  takes as input the security parameter  $\lambda$  and a boolean circuit  $\mathbf{C} : \{0, 1\}^\ell \rightarrow \{0, 1\}^m$ , and outputs a garbled circuit  $\text{GC}$  and  $\ell$  pairs of garbled labels  $\mathbf{K} = (K_1^0, K_1^1, \dots, K_\ell^0, K_\ell^1)$ . For simplicity we assume that for every  $i \in [\ell]$  and  $b \in \{0, 1\}$  it holds that  $K_\ell^b \in \{0, 1\}^\lambda$ .

$\text{eval}(\text{GC}, K_1, \dots, K_\ell) \rightarrow y$ : The evaluation algorithm  $\text{eval}$  takes as input the garbled circuit  $\text{GC}$  and  $\ell$  garbled labels  $K_1, \dots, K_\ell$ , and outputs a value  $y \in \{0, 1\}^m$ .

We require the following properties of a projective garbling scheme:

*Perfect Correctness.* We say  $\text{GC}$  satisfies perfect correctness if for any boolean circuit  $\mathbf{C} : \{0, 1\}^\ell \rightarrow \{0, 1\}^m$  and  $x = (x_1, \dots, x_\ell)$  it holds that

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

where  $(\text{GC}, \mathbf{K}) \leftarrow \text{garble}(1^\lambda, \mathbf{C})$  with  $\mathbf{K} = (K_1^0, K_1^1, \dots, K_\ell^0, K_\ell^1)$ , and  $\mathbf{K}[x] = (K_1^{x_1}, \dots, K_\ell^{x_\ell})$ .

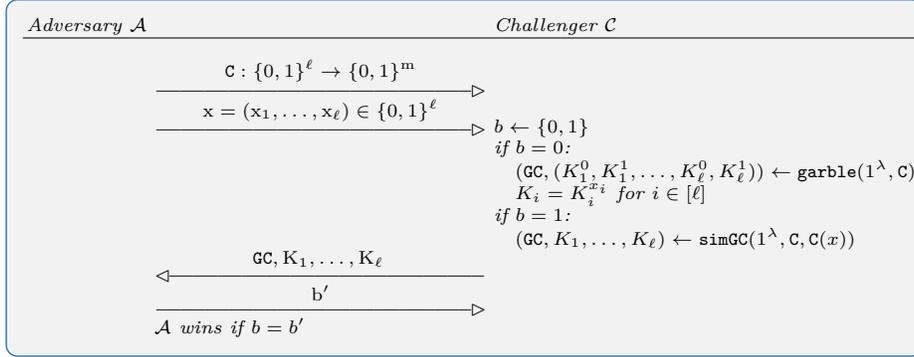
Next, we formally define the security notions we require for a garbling scheme. When garbled circuits are used in such a way that decoding information is used separately, obliviousness requires that a garbled circuit together with a set of labels reveals nothing about the input the labels correspond to, and privacy requires that the additional knowledge of the decoding information reveals only the appropriate output. In our work, we do not consider decoding information separately (but rather, consider it to be included in the garbled circuit), so we do not need obliviousness.

*Privacy.* Informally, privacy requires that a garbled circuit together with a set of labels reveal nothing about the input the labels correspond to (beyond the appropriate output).

More formally, we say that  $\text{GC}$  satisfies privacy if there exists a simulator  $\text{simGC}$  such that for every PPT adversary  $\mathcal{A}$ , it holds that

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

in the following experiment:

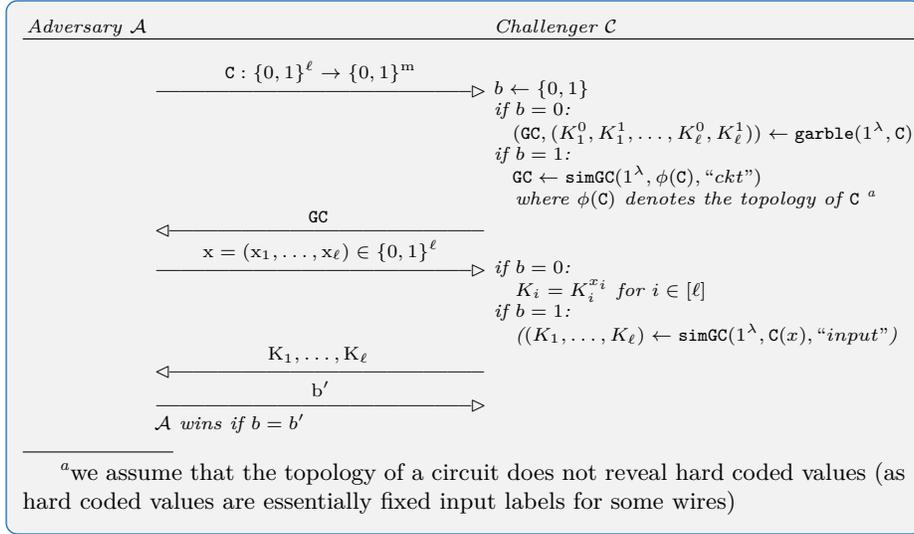


Remark. It is possible to use an alternate variant of the simulator  $\text{simGC}$  that takes as input a set of labels  $(K_1, \dots, K_\ell)$  and returns a garbled circuit  $GC$  compatible with these labels. The simulator of Yao's [Yao82] garbling scheme can be made to work easily as mentioned above.

*Adaptive Privacy.* Informally, this property requires that privacy is maintained against an adversary who first obtains the garbled circuit and then selects the input. More formally, we say that  $GC$  satisfies adaptive privacy if there exists a simulator  $\text{simGC}$  such that for every PPT adversary  $\mathcal{A}$ , it holds that

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

in the following experiment:



*Instantiation.* For our constructions, adaptive garbled circuits can be obtained using one-time pads with Yao's garbled circuits (as shown by Bellare *et al.* [BHR12a]).

## A.4 Commitment Scheme

A commitment scheme allows a party to commit to a value while keeping it hidden, and to later reveal the committed value with the guarantee that the commitment is binding [Ped92].

**Definition 10 (Commitment Scheme).** *A commitment scheme ( $\mathcal{C}$ ) is a tuple of efficient algorithms  $\mathcal{C} = (\text{com}, \text{open})$  defined as follows.*

$\text{com}(\text{msg}) \rightarrow (o, \text{com})$ : *The commitment algorithm  $\text{com}$  takes as input a message  $\text{msg} \in \{0, 1\}^\lambda$  and outputs a decommitment value  $o \in \{0, 1\}^\lambda$  and a commitment value  $\text{com} \in \{0, 1\}^\lambda$ .*

$\text{open}(o, \text{com}) \rightarrow \{\text{msg}, \perp\}$ : *The opening algorithm  $\text{open}$  takes as input a decommitment value  $o \in \{0, 1\}^\lambda$  and a commitment value  $\text{com} \in \{0, 1\}^\lambda$  and outputs either a message  $\text{msg}$  or  $\perp$  in case  $o$  is not a valid decommitment for  $\text{com}$ .*

We require the following properties of a commitment scheme:

*Correctness.* We say that a commitment scheme  $\mathcal{C}$  satisfies correctness if for any message  $\text{msg} \in \{0, 1\}^\lambda$  and  $(o, \text{com}) \leftarrow \text{com}(\text{msg})$ , we have that

$$\Pr[\text{open}(\text{com}, o) = \text{msg}] \geq 1 - \text{negl}(\lambda).$$

*Hiding.* The hiding property requires that a commitment reveals nothing about the underlying message. More formally, we say that a commitment scheme  $\mathcal{C}$  satisfies hiding if for all PPT adversaries  $\mathcal{A}$ ,

$$\Pr \left[ b = b' \left| \begin{array}{l} (\text{msg}_0, \text{msg}_1) \leftarrow \mathcal{A}(1^\lambda); \\ b \leftarrow \{0, 1\}; \\ (o, \text{com}) \leftarrow \text{com}(\text{msg}_b); \\ b' \leftarrow \mathcal{A}(\text{com}) \end{array} \right. \right] \leq \frac{1}{2} + \text{negl}(\lambda).$$

*Binding.* The binding property requires that an adversary cannot open a commitment in two different ways. More formally, we say that a commitment scheme  $\mathcal{C}$  satisfies binding if for all PPT adversaries  $\mathcal{A}$ ,

$$\Pr \left[ \text{msg}_0, \text{msg}_1 \neq \perp, \text{msg}_0 \neq \text{msg}_1 \left| \begin{array}{l} (\text{com}, o_0, o_1) \leftarrow \mathcal{A}(1^\lambda); \\ \text{msg}_0 \leftarrow \text{open}(\text{com}, o_0); \\ \text{msg}_1 \leftarrow \text{open}(\text{com}, o_1) \end{array} \right. \right] \leq \text{negl}(\lambda).$$

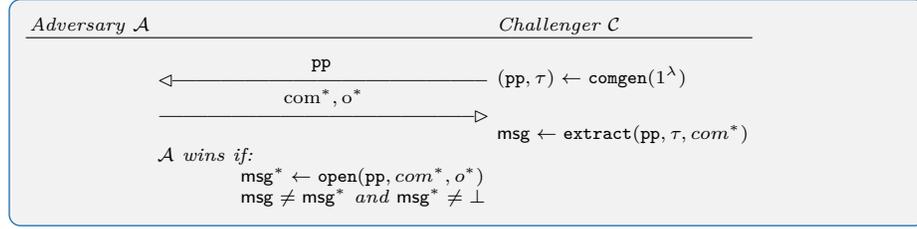
**Definition 11 (Extractable Commitment Scheme).** *An extractable commitment scheme with message space  $\{0, 1\}^\lambda$  is a commitment scheme with an additional algorithm  $\text{comgen}$ , which on input the security parameters generates public parameters  $\text{pp}$  together with a trapdoor  $\tau$ . Both  $\text{com}$  and  $\text{open}$  additionally take  $\text{pp}$  as an input.*

We require the following additional property:

*Extractability.* We say that a commitment scheme satisfies extractability if for all PPT adversaries  $\mathcal{A}$  there exists a PPT extraction algorithm  $\text{extract}$  and a negligible function  $\text{negl}(\cdot)$  such that

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

in the following experiment:



*Instantiation.* An extractable commitment scheme in the CRS model could be instantiated using, for instance, El-Gamal encryption [ELG84] (with the secret key as the trapdoor  $\tau$ ).

**Definition 12 (Equivocable commitment scheme).** An equivocable commitment scheme with message space  $\{0, 1\}^\lambda$  is a commitment scheme with three additional algorithms:  $\text{comgen}$ ,  $\text{eqcommit}_1$ , and  $\text{eqcommit}_2$ .  $\text{comgen}$  on input the security parameter generates public parameters  $\text{pp}$  together with a trapdoor  $\tau$ ; both  $\text{com}$  and  $\text{open}$  additionally take  $\text{pp}$  as an input.  $\text{eqcommit}_1$ , and  $\text{eqcommit}_2$  are as follows:

$\text{eqcommit}_1(\text{pp}, \tau) \rightarrow (\text{com}, \text{aux})$ : is the first randomized equivocation algorithm that takes as input public parameters  $\text{pp}$  and trapdoor  $\tau$  and outputs commitment  $\text{com}$  and auxiliary information  $\text{aux}$ .

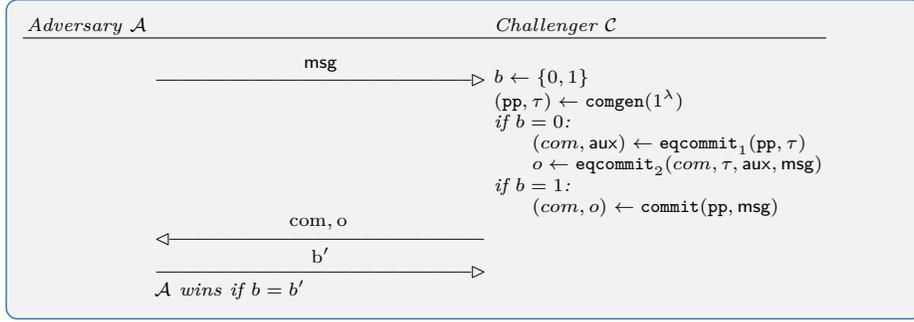
$\text{eqcommit}_2(\text{msg}, \text{com}, \tau, \text{aux}) \rightarrow o$  is the second randomized equivocation algorithm that takes as input message  $\text{msg} \in \{0, 1\}^\lambda$ , commitment  $\text{com}$ , trapdoor  $\tau$  and auxiliary information  $\text{aux}$ , and produces decommitment information  $o$  such that  $\text{open}$ , on input  $(\text{pp}, \text{com}, o)$ , outputs  $\text{msg}$ .

We require the following additional property:

*Equivocability.* An equivocal commitment scheme  $\mathcal{C}$  satisfies equivocability if for any PPT adversary  $\mathcal{A}$  there exist a negligible function  $\text{negl}(\cdot)$ , s.t.

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

in the following experiment:



*Instantiation.* An equivocal commitment scheme in the CRS model could be instantiated using, for instance, Pedersen commitments [Ped92].

### A.5 Non-Interactive Zero-Knowledge Arguments of Knowledge

We take this definition from Groth and Maller [GM17].

**Definition 13 (Non-Interactive Zero-Knowledge Arguments of Knowledge (NIZKAoK)).** A non-interactive zero-knowledge argument of knowledge (NIZK) scheme is a tuple of efficient algorithms  $\text{NIZK} = (\text{setupZK}, \text{prove}, \text{verify}, \text{simP})$  defined as follows.

$\text{setupZK}(1^\lambda, \mathcal{R}) \rightarrow (crs, td)$ : The algorithm  $\text{setupZK}$  takes as input the security parameter  $\lambda \in \mathbb{N}$ , and outputs the global common reference string  $crs$  and the trapdoor  $td$  for the NIZK system.

$\text{prove}(crs, \phi, w) \rightarrow \pi$ : The algorithm  $\text{prove}$  takes as input the common reference string  $crs$  for a relation  $\mathcal{R}$ , a statement  $\phi$  and a witness  $w$ , and outputs a proof  $\pi$  that  $(\phi, w) \in \mathcal{R}$ .

$\text{verify}(crs, \phi, \pi) \rightarrow \text{accept/reject}$ : The algorithm  $\text{verify}$  takes as input the common reference string  $crs$  for a relation  $\mathcal{R}$ , a statement  $\phi$  and a proof  $\pi$ , and verifies whether  $\pi$  proves the existence of a witness  $w$  such that  $(\phi, w) \in \mathcal{R}$ .

$\text{simP}(crs, td, \phi) \rightarrow \pi$ : The algorithm  $\text{simP}$  takes as input the common reference string  $crs$  for a relation  $\mathcal{R}$ , the trapdoor  $td$  and a statement  $\phi$ , and outputs a simulated proof of the existence of a witness  $w$  such that  $(\phi, w) \in \mathcal{R}$ .

We require the following properties of a NIZK scheme:

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

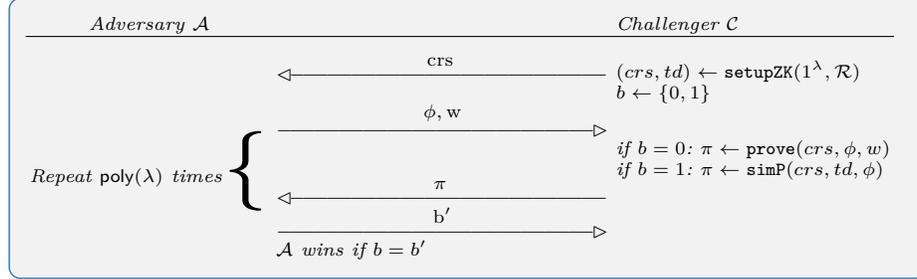
$$\Pr \left[ \text{verify}(\phi, \pi) = 1 \mid \begin{array}{l} (crs, td) \leftarrow \text{setupZK}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \text{prove}(\phi, w) \end{array} \right] \geq 1 - \text{negl}(\lambda)$$

(where the randomness is taken over the internal coin tosses of  $\text{setupZK}$ ,  $\text{prove}$  and  $\text{verify}$ ).

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

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

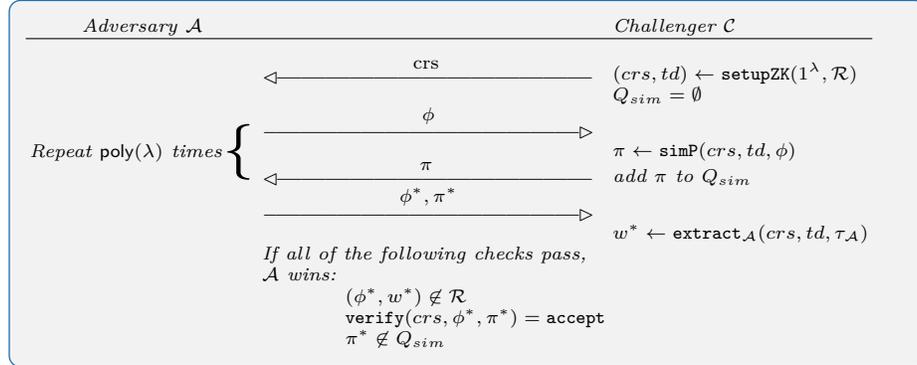
in the following experiment:



*Simulation Extractability.* We say that NIZK satisfies simulation extractability if for all PPT adversaries  $\mathcal{A}$  there exists a PPT extraction algorithm  $\text{extract}_{\mathcal{A}}$  such that

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

in the following experiment:



*Instantiation.* Simulation extractable NIZK could be instantiated using, for instance, technique from [DDO<sup>+</sup>01].

## B Non-Interactive One-or-Nothing Secret Sharing of Damgård *et al.* [DMR<sup>+</sup>21]

We recall the syntax and the construction of the non-interactive one-or-nothing secret sharing of Damgård *et al.* [DMR<sup>+</sup>21] below.

## B.1 Syntax

A non-interactive one-or-nothing secret sharing scheme consists of a tuple of four algorithms (**setup**, **share**, **vote**, **reconstruct**).

**setup**( $1^\lambda$ )  $\rightarrow$  **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 **setup** on randomness obtained from e.g. key exchange.)

**share**(**sk**<sub>1</sub>, ..., **sk**<sub>*n*</sub>, **z**<sup>(1)</sup>, ..., **z**<sup>(*l*)</sup>)  $\rightarrow$  *s* is an algorithm that takes the *n* shared keys **sk**<sub>1</sub>, ..., **sk**<sub>*n*</sub> and the *l* values **z**<sup>(1)</sup>, ..., **z**<sup>(*l*)</sup>, and produces a public share *s*.

**vote**(**sk**<sub>*i*</sub>, *v*)  $\rightarrow$   $\bar{s}_i$  is an algorithm that takes a shared key **sk**<sub>*i*</sub> 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  $\bar{s}_i$ .

**reconstruct**(*s*,  $\bar{s}_1, \dots, \bar{s}_n$ )  $\rightarrow$   $\{z^{(v)}, \perp\}$  is an algorithm that takes the public share *s*, all of the ballots  $\bar{s}_1, \dots, \bar{s}_n$ , and outputs either the value **z**<sup>(*v*)</sup> which received a majority of votes, or outputs  $\perp$ .

The properties required from a one-or-nothing secret sharing scheme are  $\delta$ -correctness, privacy and contradiction-privacy (as described informally in Section 3.2). We refer to [DMR<sup>+</sup>21] for the formal definitions.

## B.2 Generalized Construction of the Non-Interactive One-or-Nothing Secret Sharing of Damgård *et al.* [DMR<sup>+</sup>21]

We present a more generalized version of the construction of Damgård *et al.* [DMR<sup>+</sup>21] such that it satisfies  $\delta$ -correctness when  $\delta > t$ . The original construction satisfies (*n* - *t*)-correctness when  $n > 2t$ . Note that the security of this construction holds against semi-honest adversary. The tool used in the construction is a symmetric key encryption scheme (**SKE.keygen**, **enc**, **dec**) and a threshold secret sharing scheme (for the formal definitions we refer the reader to [DMR<sup>+</sup>21]).

Figure B.1: Non-Interactive One-or-Nothing Secret Sharing

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

**share**(**sk**<sub>1</sub>, ..., **sk**<sub>*n*</sub>, **z**<sup>(1)</sup>, ..., **z**<sup>(*l*)</sup>)  $\rightarrow$  *s*:

1. Compute ( $z_1^{(v)}, \dots, z_n^{(v)}$ ) as the additive sharing of **z**<sup>(*v*)</sup> for  $v \in [l]$ .
2. Compute ( $z_{i \rightarrow 1}^{(v)}, \dots, z_{i \rightarrow n}^{(v)}$ ) as the threshold sharing of  $z_i^{(v)}$  with threshold *t* for  $v \in [l], i \in [n]$ .
3. Parse ( $k_i^{(1)}, \dots, k_i^{(l)}, k_i^{(\perp)}$ ) = **sk**<sub>*i*</sub> for  $i \in [n]$ .
4. Compute  $c_i^{(v)} = \text{enc}(k_i^{(v)}, z_i^{(v)})$  for  $v \in [l], i \in [n]$ .
5. Compute  $c_{i \rightarrow j}^{(v)} = \text{enc}(k_i^{(\perp)}, \text{enc}(k_j^{(v)}, z_{i \rightarrow j}^{(v)}))$  for  $v \in [l], i \in [n], j \in [n]$ .

6. Output  $s = (\{c_i^{(v)}\}_{i \in [n], v \in [l]}, \{c_{i \rightarrow j}^{(v)}\}_{i, j \in [n], v \in [l]})$ .
- vote**( $\mathbf{sk}_i, v$ )  $\rightarrow \bar{s}_i$  where  $v \in \{1, \dots, l, \perp\}$ : Output  $\bar{s}_i = (v, \mathbf{k}_i^{(v)})$ .
- reconstruct**( $s, \bar{s}_1, \dots, \bar{s}_n$ )  $\rightarrow \{\mathbf{z}^{(v)}, \perp\}$ :
1. Parse  $(\{c_i^{(v)}\}_{i \in [n], v \in [l]}, \{c_{i \rightarrow j}^{(v)}\}_{i, j \in [n], v \in [l]}) = s$ .
  2. Parse  $(v_i, \mathbf{k}_i) = \bar{s}_i$  for  $i \in [n]$ .
  3. If there does not exist a  $v \in \{1, \dots, l\}$  such that at least  $\delta^a$  parties vote for  $v$  and everyone else votes for  $\perp$ , output  $\perp$ .
  4. Let  $v \neq \perp$  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  $\mathbf{z}_i = \text{dec}(\mathbf{k}_i, c_i^{(v)})$ .
  6. For  $i \notin S$  (so,  $v_i = \perp$ ), for each  $j \in S$ , compute  $\mathbf{z}_{i \rightarrow j} = \text{dec}(\mathbf{k}_i, \text{dec}(\mathbf{k}_j, c_{i \rightarrow j}^{(v)}))$ . Let  $\mathbf{z}_i$  denote the value reconstructed using the threshold shares  $\{\mathbf{z}_{i \rightarrow j}\}_{j \in S}$ .
  7. If there exists any  $i$  such that  $\mathbf{z}_i = \perp$ , output  $\perp$ . Else, output  $\mathbf{z} = \sum_{i=1}^n \mathbf{z}_i$ .

---

<sup>a</sup>this is the only modification to the construction in [DMR<sup>+</sup>21] which uses  $\delta = n - t$ .

The above construction satisfies  $\delta$ -correctness, privacy and contradiction-privacy, when  $\delta > t$  as shown below. The proof of privacy and contradiction-privacy is verbatim from [DMR<sup>+</sup>21] as it remains unaffected by our modification.

**Correctness:** Suppose a set of at least  $\delta$  parties vote for  $v$ , and others vote for  $\perp$ . Let  $S$  and  $T$  denote the set of indices corresponding to parties voting for  $v$  and  $\perp$  respectively. Firstly, it directly follows from the steps of **reconstruct** that  $\mathbf{z}_i^{(v)} \neq \perp$  can be obtained for  $i \in S$ . Further, since  $|S| \geq \delta > t$  holds, it follows that  $\mathbf{z}_i^{(v)}$  shared using threshold  $t$  can be successfully reconstructed corresponding to each  $i \in T$ . Lastly, since  $S \cup T = [n]$ , we can infer that  $\mathbf{z}_i^{(v)} \neq \perp$  for each  $i \in [n]$ ; thereby **reconstruct** would output  $\mathbf{z}^{(v)}$ .

**Privacy:** Suppose no honest party votes for  $v$ . Consider an honest party  $i$ . We show that the adversary learns nothing about  $\mathbf{z}_i^{(v)}$ , which suffices to show that  $\mathbf{z}^{(v)}$  remains perfectly hidden from the adversary. If party  $i$  votes for  $v_i \neq v$ , then it follows from the steps in **vote** that party  $i$  does not reveal any information related to  $\mathbf{k}_i^{(v)}$  or  $\mathbf{k}_i^{(\perp)}$ . Due to the security of the encryption scheme, the adversary learns nothing about  $\mathbf{z}_i^{(v)}$  from  $c_i^{(v)}$  and  $\{c_{i \rightarrow j}^{(v)}\}_{j \in [n]}$ . Next, suppose party  $i$  votes for  $\perp$  and publishes  $\mathbf{k}_i^{(\perp)}$ . In this case, we note that the adversary can use  $\mathbf{k}_i^{(\perp)}$  and  $\mathbf{k}_j^{(v)}$  to decrypt  $c_{i \rightarrow j}^{(v)}$  and obtain the share  $\mathbf{z}_{i \rightarrow j}^{(v)}$  if party  $j$  is corrupt. However,  $c_{i \rightarrow j}^{(v)}$  corresponding to an honest party  $j$  can be decrypted by the adversary only if party  $j$  reveals  $\mathbf{k}_j^{(v)}$ ; since this occurs only when honest party  $j$  votes for  $v$  (which does not happen as per our assumption), we can conclude that the adversary learns no information from  $c_{i \rightarrow j}^{(v)}$  corresponding to an honest party  $j$ . Thus, the adversary has

access to at most  $t$  shares of  $\mathbf{z}_i^{(v)}$  which is shared using threshold  $t$ . It now follows from the privacy of threshold secret sharing that the adversary learns no information about  $\mathbf{z}_i^{(v)}$ .

**Contradiction Privacy:** Suppose two different honest parties, say parties  $i$  and  $j$ , publish votes for  $v_i \neq v_j$ ,  $v_i, v_j \neq \perp$ . First, we argue that the adversary learns nothing about  $\mathbf{z}_j^{(v_i)}$ , which suffices to prove that adversary learns nothing about  $\mathbf{z}^{(v_i)}$ . Since honest party  $j$  voted for  $v_j \neq v_i$ , the adversary does not have access to  $\mathbf{k}_j^{(v_i)}$  or  $\mathbf{k}_j^{(\perp)}$ . Therefore, the adversary cannot obtain any information related to  $\mathbf{z}_j^{(v_i)}$  via  $c_j^{(v_i)}$  or  $c_{j \rightarrow k}^{(v_i)}$  (for any  $j \in [n]$ ). A similar argument as above can be used to show that the adversary learns nothing about  $\mathbf{z}_i^{(v_j)}$ , which suffices to prove that adversary learns nothing about  $\mathbf{z}^{(v_j)}$ .

## C One-or-Nothing Secret Sharing with Intermediaries of Damgård *et al.* [DRSY23]

We recall the syntax and the construction of the maliciously-secure one-or-nothing secret sharing with intermediaries of Damgård *et al.* [DRSY23] below.

### C.1 Syntax

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

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

$\text{share}(crs, \mathbf{pk}_1, \dots, \mathbf{pk}_n, \mathbf{z}^{(1)}, \dots, \mathbf{z}^{(l)}) \rightarrow s$  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  $s$ .

$\text{vote}(crs, \mathbf{sk}_i, \mathbf{pk}_1, \dots, \mathbf{pk}_n, v_i) \rightarrow \bar{s}_i$  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  $v_i \in \{1, \dots, l, \perp\}$  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 returns a ballot  $\bar{s}_i$ .

Note that, to allow  $\text{share}$  and  $\text{vote}$  to be executed in a single round,  $\text{vote}$  does not take as input the share  $s$ .

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

The properties required from a one-or-nothing secret sharing with intermediaries are  $\epsilon$ -privacy,  $\alpha$ -identifiability and  $\beta$ -correctness, (as described informally in Section 3.2). The definitions assume that corrupt parties might provide honest parties, including the dealer, with inconsistent or incorrect public keys. We refer to [DRSY23] for the formal definitions.

## C.2 (Modified) Construction of the One-or-Nothing Secret Sharing with Intermediaries of Damgård *et al.* [DRSY23]

The maliciously-secure construction uses the tools of a public key encryption scheme with CPA security, say  $\text{PKE} = (\text{keygen}, \text{enc}, \text{dec})$  and a non-interactive zero-knowledge proof system  $\text{NIZK} = (\text{setupZK}, \text{prove}, \text{verify}, \text{simP}, \text{extract})$  (Section A.5) for the following relations:

$$\begin{aligned} \mathcal{R}_{\text{keygen}} &= \left\{ \phi = \text{pk} \right. \\ &\quad \left. w = (\text{sk}, r) \mid (\text{sk}, \text{pk}) \leftarrow \text{PKE.keygen}(1^\lambda; r) \right\}, \\ \mathcal{R}_{\text{share}} &= \left\{ \phi = \left\{ \text{pk}_{i \rightarrow j}^{(v)}, c_{i \rightarrow j}^{(v)} \right\}_{v \in [l], i, j \in [n]} \right. \\ &\quad \left. w = \left( \begin{array}{l} \{z^{(v)}, r^{(v)}, \{r_i^{(v)}, \\ \{r_{i \rightarrow j}^{(v)}\}_{j \in [n]}\}_{i \in [n]}\}_{v \in [l]} \end{array} \right) \mid \left\{ \begin{array}{l} \{(s_1^{(v)}, \dots, s_n^{(v)}) \leftarrow \text{Shamir.share}(z^{(v)}; r^{(v)})\}_{v \in [l]} \\ \{(s_{i \rightarrow 1}^{(v)}, \dots, s_{i \rightarrow n}^{(v)}) \leftarrow \text{Shamir.share}(s_i^{(v)}; r_i^{(v)})\}_{v \in [l], i \in [n]} \\ \wedge \{c_{i \rightarrow j}^{(v)} \leftarrow \text{PKE.enc}(\text{pk}_{i \rightarrow j}^{(v)}, s_{i \rightarrow j}^{(v)}; r_{i \rightarrow j}^{(v)})\}_{v \in [l], i, j \in [n]} \end{array} \right\} \right. \\ \mathcal{R}_{\text{vote}} &= \left\{ \phi = \left( \begin{array}{l} \{\text{pk}_{j \rightarrow j}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}, \text{tk}_{j \rightarrow i}^{(v)}\}_{v \in [l], j \in [n]}, \\ v_i, \text{sk}_{i \rightarrow i}^{(v_i)} \end{array} \right) \right. \\ &\quad \left. w = \left( \begin{array}{l} \{\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}, r_j^{(v)}\}_{v \in [l], j \in [n]} \end{array} \right) \mid \left\{ \begin{array}{l} \{(\text{sk}_{j \rightarrow i}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}) \leftarrow \text{PKE.keygen}(1^\lambda; \bar{r}_{j \rightarrow i}^{(v)})\}_{j \in [n], v \in [l]} \\ \wedge \{\text{tk}_{j \rightarrow i}^{(v)} \leftarrow \text{PKE.enc}(\text{pk}_{j \rightarrow j}^{(v)}, \text{sk}_{j \rightarrow i}^{(v)}; r_j^{(v)})\}_{v \in [l], j \in [n]} \end{array} \right\} \right. \end{aligned}$$

We refer the reader to [DRSY23] for the formal definitions of these tools.

Figure C.1 describes the modified version of the one-or-nothing secret sharing with intermediaries (1or0wi) scheme of Damgård *et al.* [DRSY23] achieving the properties outlined in Theorem 9. The minor tweaks to satisfy the properties outlined in Theorem 10 are marked in **blue**.

Figure C.1: Construction of 1or0wi
<p><b>setup</b>(<math>1^\lambda</math>): Set up and output the common reference strings</p> $\begin{aligned} crs_{\text{keygen}} &\leftarrow \text{setupZK}(1^\lambda, \mathcal{R}_{\text{keygen}}), \\ crs_{\text{share}} &\leftarrow \text{setupZK}(1^\lambda, \mathcal{R}_{\text{share}}), \text{ and} \\ crs_{\text{vote}} &\leftarrow \text{setupZK}(1^\lambda, \mathcal{R}_{\text{vote}}) \end{aligned}$ <p>for the zero knowledge proof system. Return <math>crs = (crs_{\text{keygen}}, crs_{\text{share}}, crs_{\text{vote}})</math>.</p> <p><b>keygen</b>(<math>crs</math>), <b>run by party <math>i</math></b>:</p> <ol style="list-style-type: none"> <li>1. For each <math>j \in [n]</math> and <math>v \in [l]</math>, <math>(\text{sk}_{j \rightarrow i}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}) \leftarrow \text{PKE.keygen}(1^\lambda; \bar{r}_{j \rightarrow i}^{(v)})</math>.</li> <li>2. For each <math>j \in [n]</math> and <math>v \in [l]</math>, <math>\pi_{j \rightarrow i}^{(v)} \leftarrow \text{NIZK.prove}(crs_{\text{keygen}}, \phi = \text{pk}_{j \rightarrow i}^{(v)}, w = (\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}))</math>.</li> <li>3. Let <math>\text{sk}_i = (\{\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]})</math>, and <math>\text{pk}_i = (\{\text{pk}_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]})</math>.</li> <li>4. Output <math>(\text{sk}_i, \text{pk}_i)</math>.</li> </ol> <p><b>share</b>(<math>crs, \text{pk}_1, \dots, \text{pk}_n, z^{(1)}, \dots, z^{(l)}</math>), <b>run by the dealer <math>D</math></b> (where <math>\text{pk}_i = \{\text{pk}_{j \rightarrow i}^{(v)}, \pi_{j \rightarrow i}^{(v)}\}_{j \in [n], v \in [l]}</math>):</p>

1. For each  $v \in [l]$ , compute  $(s_1^{(v)}, \dots, s_n^{(v)}) \leftarrow \text{Shamir.share}(\mathbf{z}^{(v)}; r^{(v)})$  as the threshold sharing of  $\mathbf{z}^{(v)}$  with threshold  $t_1 = n - t - t_m - 1$  (**resp.**,  $t_1 = n - t_m - 1$ ).
2. For each  $i \in [n]$  and  $v \in [l]$ , compute  $(s_{i \rightarrow 1}^{(v)}, \dots, s_{i \rightarrow n}^{(v)}) \leftarrow \text{Shamir.share}(s_i^{(v)}; r_i^{(v)})$  as the threshold sharing of  $s_i^{(v)}$  with threshold  $t_2 = n - 2t - t_m - t_d - 1$  (**resp.**,  $t_2 = n - t_m - t_d - 1$ ).
3. For each  $i, j \in [n]$  and  $v \in [l]$ , compute  $c_{i \rightarrow j}^{(v)} \leftarrow \text{PKE.enc}(\text{pk}_{i \rightarrow j}^{(v)}, s_{i \rightarrow j}^{(v)}; r_{i \rightarrow j}^{(v)})$ .
4. Set
  - $\phi_{\text{share}} = (\{\text{pk}_{i \rightarrow j}^{(v)}, c_{i \rightarrow j}^{(v)}\}_{v \in [l], i, j \in [n]})$  and
  - $w_{\text{share}} = (\{\mathbf{z}^{(v)}, r^{(v)}, \{r_i^{(v)}, \{r_{i \rightarrow j}^{(v)}\}_{j \in [n]}\}_{i \in [n]}\}_{v \in [l]})$ .
 Compute  $\pi_{\text{share}} \leftarrow \text{prove}(crs_{\text{share}}, \phi_{\text{share}}, w_{\text{share}})$ .
5. Set  $s = (\phi_{\text{share}}, \pi_{\text{share}})$  and output  $s$ .

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

1. For each  $v \in [l]$  and  $j \in [n]$ , let  $\text{tk}_{j \rightarrow i}^{(v)} \leftarrow \text{PKE.enc}(\text{pk}_{j \rightarrow j}^{(v)}, \text{sk}_{j \rightarrow i}^{(v)}; r_j^{(v)})$ .
2. Set
  - $\phi_{\text{vote}, i} = (\{\text{pk}_{j \rightarrow j}^{(v)}, \text{pk}_{j \rightarrow i}^{(v)}, \text{tk}_{j \rightarrow i}^{(v)}\}_{v \in [l], j \in [n]}, v_i, \text{sk}_{i \rightarrow i}^{(v_i)})^a$
  - $w_{\text{vote}, i} = (\{\text{sk}_{j \rightarrow i}^{(v)}, \bar{r}_{j \rightarrow i}^{(v)}, r_j^{(v)}\}_{v \in [l], j \in [n]})$ .
 Compute  $\pi_{\text{vote}, i} \leftarrow \text{prove}(crs_{\text{vote}}, \phi_{\text{vote}, i}, w_{\text{vote}, i})$ .
3. Set  $\bar{s}_i = (\phi_{\text{vote}, i}, \pi_{\text{vote}, i})$  and output  $\bar{s}_i$ .

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

Identify the winning vote:

1. If there does not exist a  $v \in \{1, \dots, l\}$  such that at least  $t_1$  parties vote for  $v$ , output  $\perp$ . Let  $S_{\text{vote}} \subseteq [n]$  be the set of parties  $i$  such that  $v_i = v$ .

Verify the zero knowledge proofs:

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

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  $\text{pk}_{i \rightarrow i}^{(v)}$  is the same in  $\text{pk}_i$  and  $\bar{s}_j$ .<sup>b</sup> If  $|S'_i| < n - t - t_m$ , return  $\perp_i$  (**resp.**, if  $|S'_i| < n - t_m$ , return  $\perp$ ).

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

---

*Maliciously secure one-or-nothing secret sharing with intermediaries*

<sup>a</sup>any string  $m^{(\perp)}$  is to be interpreted as  $\perp$ .

<sup>b</sup>This is the crucial step that differs from the original construction where it is additionally checked that parties  $P_i$  and  $P_j$  are in agreement with respect to  $\text{pk}_{j \rightarrow i}^{(v)}$  as well.

*Proof of Theorem 9* We argue below that the above construction satisfies  $(n - t_m - t)$ -identifiability and  $(n - 2t - t_m)$ -privacy when  $n > 3t + t_m + t_d$  and  $n > 3t + 2t_m$  holds.

**$(n - t_m - t)$ -Identifiability.** Suppose  $(n - t_m - t)$  parties, say constituting the set  $S_{\text{vote}}$ , produce ballot using the same  $v$ . Let  $\mathbb{H}$  denote the set of indices corresponding to honest parties.

First, we argue that  $\perp_i$ , where  $i \in \mathbb{H}$  is output with negligible probability. We observe that this can happen only when one of the following holds **(a)** NIZK proof  $\pi_{\text{share}}$  (if  $i$  is the dealer) or  $\pi_{\text{vote}, i}$  or  $\pi_{j \rightarrow i}^{(v)}$  (for any  $j \in [n]$ ) does not verify or **(b)**  $|S'_i| < n - t - t_m$  or **(c)**  $i$  is the dealer and  $|S_D| < n - t - t_d$ . First, it directly follows from correctness of NIZK that **(a)** cannot occur with respect to an honest  $P_i$ , except with negligible probability. Next, we note that  $|S'_i| \geq n - t - t_m$  must hold as an honest  $P_i$ 's public keys will be received by at least  $n - t_m$  parties in the first round; and in the worst case  $t$  among them are corrupt and do not echo  $P_i$ 's public keys. We can thus infer that **(b)** cannot hold. Further, the set of  $n - t_d$  parties from whom an honest dealer should hear in the first round may constitute at most  $t$  corrupt parties in the worst case. These corrupt parties may not communicate to the honest dealer in the first round or may possibly broadcast different public keys in the second round (i.e. different than the one they sent to the dealer in the first round). Thus, we can conclude that the public keys broadcast by the honest dealer must be consistent with the public keys of at least  $n - t_d - t$  parties. Thus,  $|S_D| \geq n - t - t_d$ , implying that **(c)** cannot hold. This completes the argument that  $\perp_i$ , where  $i \in \mathbb{H}$  is output with negligible probability. From the above, we can conclude that when  $\perp_i$  is

output,  $i \notin \mathbb{H}$  with overwhelming probability. Therefore, to complete the proof, it suffices to show that when no cheater is identified,  $\mathbf{z}^{(v)}$  is reconstructed.

Suppose no cheating party is identified. Since  $|S_D| \geq n - t - t_d > 2t + t_m$  must hold,  $S_D$  constitutes honest parties who must have verified  $\pi_{\text{share}}$  by a potentially corrupt dealer. It follows from simulation-soundness of NIZK that the encryptions  $\{c_{k \rightarrow j}^{(v)}\}_{k,j \in [n]}$  must have indeed been computed correctly. Next, consider  $k \in S_{\text{vote}}$  and  $j \in S_k$  (where  $S_k = S'_k \cap S_D$  and  $|S_k| \geq n - 2t - t_m - t_d$ ). Recall that  $P_k$  and  $P_j$  are in agreement with respect to the public key  $\text{pk}_{k \rightarrow k}^{(v)}$  and  $(\pi_{\text{vote},k}, \pi_{k \rightarrow k}^{(v)})$  sent by  $P_k$  and  $(\pi_{\text{vote},j}, \pi_{k \rightarrow j}^{(v)})$  sent by  $P_j$  verified successfully (otherwise a party must have been identified as cheater). Simulation soundness of NIZK ensures that the public key  $\text{pk}_{k \rightarrow k}^{(v)}$  used by  $P_j$  to broadcast ciphertext  $\text{tk}_{k \rightarrow j}^{(v)}$  (contained in  $\bar{s}_j$ ) corresponds to the secret key  $\text{sk}_{k \rightarrow k}^{(v)}$  (contained in  $\bar{s}_k$ ) broadcast by  $P_k$ . It now follows from the correctness of the encryption scheme that  $\text{sk}_{k \rightarrow j}^{(v)}$  is obtained upon decrypting  $\text{tk}_{k \rightarrow j}^{(v)}$  with overwhelming probability. Further, since  $j \in S_D$ , it follows from simulation-soundness of the NIZK  $\pi_{\text{share}}$  and  $\pi_{\text{vote},j}$  that the public key  $\text{pk}_{k \rightarrow j}^{(v)}$  (used by dealer for the ciphertext  $c_{k \rightarrow j}^{(v)}$ ) corresponds to the secret key  $\text{sk}_{k \rightarrow j}^{(v)}$  (used by  $P_j$  during `vote`). It thus follows  $s_{k \rightarrow j}^{(v)}$  is obtained upon decrypting  $c_{k \rightarrow j}^{(v)}$  with overwhelming probability.

Next, since  $|S_k| \geq n - 2t - t_m - t_d$  and  $s_k^{(v)}$  is shared using threshold  $(n - 2t - t_m - t_d - 1)$ , it follows from correctness of shamir's threshold sharing that  $s_k^{(v)} \neq \perp$  is reconstructed successfully for all  $k \in S_{\text{vote}}$ . Lastly, since  $|S_{\text{vote}}| \geq n - t - t_m$ ,  $\mathbf{z}^{(v)} \neq \perp$  which is shared using threshold  $(n - t - t_m - 1)$  is also reconstructed successfully (due to correctness of shamir's threshold sharing). This completes the proof.

**$(n - 2t - t_m)$ -Privacy.** Let  $\mathbb{H}$  denote the set of indices of honest parties and  $S \subset \mathbb{H}$  (where  $|S| \leq n - 2t - t_m - 1$ ) denote the set of indices of honest parties that produce ballot for  $v$ . We show that the adversary learns nothing about  $s_i^{(v)}$  for any  $i \in \mathbb{H} \setminus S$ . This would suffice to show that the adversary learns nothing about  $\mathbf{z}^{(v)}$ . This is because the adversary would have access to at most  $t + |S| \leq t + (n - 2t - t_m - 1) = n - t - t_m - 1$  shares of  $\mathbf{z}^{(v)}$  (which is shared using threshold  $(n - t - t_m - 1)$ ). Privacy of threshold sharing dictates that the adversary learns nothing about  $\mathbf{z}^{(v)}$ .

Consider an honest party  $i \in \mathbb{H} \setminus S$ . Suppose party  $i$  votes for  $v_i \neq v$ . Firstly, we argue that the adversary learns nothing about  $\text{sk}_{i \rightarrow j}^{(v)}$  for any  $j \in \mathbb{H}$  as follows : Based on the specifications of `vote`, honest  $P_i$  reveals nothing about  $\text{sk}_{i \rightarrow i}^{(v)}$ . It now follows from the CPA security of the PKE that the adversary learns nothing about  $\text{sk}_{i \rightarrow j}^{(v)}$  from  $\text{tk}_{i \rightarrow j}^{(v)}$  (contained in  $\bar{s}_j$ ). Further, the zero-knowledge property of NIZK generated by honest  $P_j$  ensures that the adversary learns nothing about  $\text{sk}_{i \rightarrow j}^{(v)}$  from  $\{\pi_{i \rightarrow j}^{(v)}\}$  (contained in  $\text{pk}_j$ ) and  $\pi_{\text{vote},j}$ . We can now infer from CPA security of the PKE and zero-knowledge property of the NIZK generated by the honest dealer, that the adversary learns nothing about  $\{s_{i \rightarrow j}^{(v)}\}_{j \in \mathbb{H}}$  from  $\{c_{i \rightarrow j}^{(v)}\}_{j \in \mathbb{H}}$  and  $\pi_{\text{share}}$  respectively.

We can thus conclude that the adversary has access to at most  $t$  shares of  $s_i^{(v)}$  which is shared using threshold  $(n - 2t - t_m - t_d - 1) \geq t$ . It now follows from privacy of threshold sharing that the adversary learns nothing about  $s_i^{(v)}$ ; completing the proof.

Lastly, we point that  $(n - 2t - t_m)$ -Privacy implies that at most one secret is reconstructed when  $n > 3t + 2t_m$  holds. To see this, suppose secrets at two different indices, say  $v$  and  $v'$  are learnt by the adversary. Then, privacy dictates that at least  $(n - 2t - t_m)$  honest parties produced ballots for  $v$  and a disjoint set of at least  $(n - 2t - t_m)$  honest parties produced ballots for  $v'$ . This can occur only if  $(n - 2t - t_m) + (n - 2t - t_m) \leq n - t$  holds which contradicts the assumption that  $n > 3t + 2t_m$ .

*Proof of Theorem 10* We argue below that the above construction (with the tweaks in [blue](#)) satisfies  $(n - t_m)$ -correctness and  $(n - t - t_m)$ -privacy when  $n > t + t_m + t_d$  and  $n > t + 2t_m$  holds.

**$(n - t_m)$ -Correctness.** Suppose everyone behaves honestly. Then,  $|S_D| \geq n - t_d$  must hold as the dealer must have received public keys in the first round from at least  $n - t_d$  parties, all of whom would echo the same public key in the second round. Further,  $|S'_k| \geq n - t_m$  must hold for each  $k \in [n]$  as the public keys  $\mathbf{pk}_{k \rightarrow k}^{(v)}$  sent by  $P_k$  in the first round must be received by at least  $n - t_m$  parties, all of whom would echo the same public key in the second round. Let  $S_{\text{vote}}$  (where  $|S_{\text{vote}}| \geq n - t_m$ ) denote the set of parties that produce the ballot using the same  $v$  and rest produce the ballot using  $\perp$ . Next, consider  $k \in S_{\text{vote}}$  and  $j \in S_k$  (where  $S_k = S'_k \cap S_D$  and  $|S_k| \geq n - t_m - t_d$ ). The public key  $\mathbf{pk}_{k \rightarrow k}^{(v)}$  used by  $P_j$  to broadcast ciphertext  $\mathbf{tk}_{k \rightarrow j}^{(v)}$  (contained in  $\bar{s}_j$ ) corresponds to the secret key  $\mathbf{sk}_{k \rightarrow k}^{(v)}$  (contained in  $\bar{s}_k$ ) broadcast by  $P_k$ . It now follows from the correctness of the encryption scheme that  $\mathbf{sk}_{k \rightarrow j}^{(v)}$  is obtained upon decrypting  $\mathbf{tk}_{k \rightarrow j}^{(v)}$  with overwhelming probability. Further, since the public key  $\mathbf{pk}_{k \rightarrow j}^{(v)}$  (used by dealer for the ciphertext  $c_{k \rightarrow j}^{(v)}$ ) corresponds to the secret key  $\mathbf{sk}_{k \rightarrow j}^{(v)}$  (used by  $P_j$  during **vote**). It thus follows  $s_{k \rightarrow j}^{(v)}$  is obtained upon decrypting  $c_{k \rightarrow j}^{(v)}$  with overwhelming probability.

Next, since  $|S_k| \geq n - t_m - t_d$  and  $s_k^{(v)}$  is shared using threshold  $n - t_m - t_d - 1$ , it follows from correctness of shamir's threshold sharing that  $s_k^{(v)} \neq \perp$  is reconstructed successfully for all  $k \in S_{\text{vote}}$ . Lastly, since  $|S_{\text{vote}}| \geq n - t_m$ ,  $\mathbf{z}^{(v)} \neq \perp$  which is shared using threshold  $(n - t_m - 1)$  is also reconstructed successfully (due to correctness of shamir's threshold sharing). This completes the proof.

**$(n - t - t_m)$ -Privacy.** Let  $\mathbb{H}$  denote the set of indices of honest parties and  $S \subset \mathbb{H}$  (where  $|S| \leq n - t - t_m - 1$ ) denote the set of indices of honest parties that produce ballot for  $v$ . We show that the adversary learns nothing about  $s_i^{(v)}$  for any  $i \in \mathbb{H} \setminus S$ . This would suffice to show that the adversary learns nothing about  $\mathbf{z}^{(v)}$ . This is because the adversary would have access to at most  $t + |S| \leq t + (n - t - t_m - 1) = n - t_m - 1$  shares of  $\mathbf{z}^{(v)}$  (which is shared using

threshold  $(n - t_m - 1)$ ). Privacy of threshold sharing dictates that the adversary learns nothing about  $\mathbf{z}^{(v)}$ .

Consider an honest party  $i \in \mathbb{H} \setminus S$ . Suppose party  $i$  votes for  $v_i \neq v$ . Firstly, we argue that the adversary learns nothing about  $\mathbf{sk}_{i \rightarrow j}^{(v)}$  for any  $j \in \mathbb{H}$  as follows : Based on the specifications of `vote`, honest  $P_i$  reveals nothing about  $\mathbf{sk}_{i \rightarrow i}^{(v)}$ . It now follows from the CPA security of the PKE that the adversary learns nothing about  $\mathbf{sk}_{i \rightarrow j}^{(v)}$  from  $\mathbf{tk}_{i \rightarrow j}^{(v)}$  (contained in  $\bar{s}_j$ ). Further, the zero-knowledge property of NIZK generated by honest  $P_j$  ensures that the adversary learns nothing about  $\mathbf{sk}_{i \rightarrow j}^{(v)}$  from  $\{\pi_{i \rightarrow j}^{(v)}\}$  (contained in  $\mathbf{pk}_j$ ) and  $\pi_{\text{vote},j}$ . We can now infer from CPA security of the PKE and zero-knowledge property of the NIZK generated by the honest dealer, that the adversary learns nothing about  $\{s_{i \rightarrow j}^{(v)}\}_{j \in \mathbb{H}}$  from  $\{c_{i \rightarrow j}^{(v)}\}_{j \in \mathbb{H}}$  and  $\pi_{\text{share}}$  respectively.

We can thus conclude that the adversary has access to at most  $t$  shares of  $s_i^{(v)}$  which is shared using threshold  $(n - t_d - t_m - 1) \geq t$ . It now follows from privacy of threshold sharing that the adversary learns nothing about  $s_i^{(v)}$ ; completing the proof.

Lastly, we point that  $(n - t - t_m)$ -Privacy implies that at most one secret is reconstructed when  $n > t + 2t_m$ . To see this, suppose secrets at two different indices, say  $v$  and  $v'$  are learnt by the adversary. Then, privacy dictates that at least  $(n - t - t_m)$  honest parties produced ballots for  $v$  and a disjoint set of at least  $(n - t - t_m)$  honest parties produced ballots for  $v'$ . This can occur only if  $(n - t - t_m) + (n - t - t_m) \leq n - t$  holds which contradicts the assumption that  $n > t + 2t_m$ .

## D Formal description of $\Pi_{\text{sl-abort}}$

In this section, we describe the BC-P2P, SA, CRS protocol  $\Pi_{\text{sl-abort}}$ , assuming  $n > 2t$  (used in Theorem 20). This protocol is a modified version of the protocol of Ananth *et al.* [ACGJ18]. We assume that the parties have access to the following tools

### Tools.

- A two-round broadcast protocol  $\Pi_{\text{bc}}$  achieving security with unanimous abort, which can be instantiated from existing protocols [BL18][GS18].  $\Pi_{\text{bc}}$  is represented by the set of functions  $\{\text{frst-msg}_i, \text{snd-msg}_i, \text{output}_i\}_{i \in [n]}$ .
- An adaptive garbling scheme (`garble`, `eval`, `simGC`) (Definition 9).
- The Shamir threshold secret sharing scheme (`Shamir.share`, `Shamir.reconstruct`, `Shamir.simshare`) (Definition 8).
- A non-interactive equivocal commitment scheme (`comgen`, `commit`, `eqcommit1`, `eqcommit2`, `open`) (Definition 12)
- A non-interactive simulation sound zero-knowledge proof system (`setupZK`, `prove`, `verify`, `simP`, `extract`) (Definition 13), for the following relation:

$$\mathcal{R} = \left\{ \begin{array}{l} \phi = \left( \begin{array}{l} i \in \{1, \dots, n\}, \text{msg}_i, \text{GC} \\ \{com_{j,l}^{(b)}, \text{pp}\}_{j \in [n], b \in \{0,1\}, l \in [L]} \end{array} \right) \\ w = \left( \begin{array}{l} x, r, \mathbf{C}_{i,x,r}, R_1, R_2, \\ \{o_{j,l}^{(b)}\}_{j \in [n], b \in \{0,1\}, l \in [L]}, \\ \{K_{j,l}^{(b)}\}_{j \in [n], b \in \{0,1\}, l \in [L]} \end{array} \right) \end{array} \middle| \begin{array}{l} \text{msg}_i \leftarrow \text{frst-msg}_i(x, r) \\ \wedge (\text{GC}, \{K_l^{(b)}\}_{b \in \{0,1\}, l \in [L]}) \leftarrow \text{garble}(1^\lambda, \mathbf{C}_{i,x,r}; R_1) \\ \wedge \text{For } b \in \{0,1\}, l \in [L] : \\ \quad (K_{1,l}^{(b)}, \dots, K_{n,l}^{(b)}) \leftarrow \text{Shamir.share}(K_l^{(b)}; R_2) \\ \wedge \text{For } b \in \{0,1\}, l \in [L], j \in [n] : \\ \quad K_{j,l}^{(b)} \leftarrow \text{open}(\text{pp}, com_{j,l}^{(b)}, o_{j,l}^{(b)}). \end{array} \right\},$$

**Notation.** Let  $\mathbf{C}_{i,x_i,r_i}(\text{msg}_1^1, \dots, \text{msg}_n^1)$  denote the boolean circuit that has  $P_i$ 's input  $x_i$  and randomness  $r_i$  hardcoded inside it, and takes as input the first round messages  $\text{msg}_1^1, \dots, \text{msg}_n^1$ , and outputs  $\text{msg}_i^2$ . For simplicity assume that each first round message is  $\ell$  bits long, so each circuit has  $L = n \cdot \ell$  input bits. Note that  $\mathbf{C}_i$  is public. Let  $g$  be the size of a garbled  $\mathbf{C}_i$ .

Figure D.1:  $\Pi_{\text{sl-abort}}$  with  $n > 2t$

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

**Setup.**

- $crs \leftarrow \text{setupZK}(1^\lambda, \mathcal{R})$ .
- $\text{pp}, \tau \leftarrow \text{comgen}(1^\lambda)$ .
- The CRS is set to  $crs, \text{pp}$ .

**First Round.** Each party  $P_i$  does the following:

1. Let  $\text{msg}_i^1 \leftarrow \text{frst-msg}_i(x_i, r_i)$  be  $P_i$ 's first round message in  $\Pi_{\text{bc}}$ .
2. Compute  $(\text{GC}_i, \mathbf{K}_i) \leftarrow \text{garble}(1^\lambda, \mathbf{C}_{i,x_i,r_i}; R_{i,1})$ , where  $\mathbf{K}_i = \{K_{i,l}^{(0)}, K_{i,l}^{(1)}\}_{l \in [L]}$ .
3. Compute threshold sharing of the input labels of  $\text{GC}_i$  with threshold  $t$  (where  $t < n/2$ ) as follows: For each  $l \in [L], b \in \{0,1\}$ , let  $(K_{1,i,l}^{(b)}, \dots, K_{n,i,l}^{(b)}) \leftarrow \text{Shamir.share}(K_{i,l}^{(b)}, R_{i,2})$ .
4. Commit to the shares of the input labels using a non-interactive commitment scheme: For each  $l \in [L], b \in \{0,1\}, j \in [n]$ , compute  $(com_{j,i,l}^{(b)}, o_{j,i,l}^{(b)}) \leftarrow \text{commit}(\text{pp}, K_{j,i,l}^{(b)})$ . Set  $com_i = \{com_{j,i,l}^{(b)}\}_{j \in [n], b \in \{0,1\}, l \in [L]}$ .
5. Set  $\phi_i = (i, \text{msg}_i, \text{GC}_i, com_i)$  and  $w_i = (x_i, r_i, \mathbf{C}_{i,x_i,r_i}, R_{i,1}, R_{i,2}, \{o_{j,i,l}^{(b)}\}_{j \in [n], b \in \{0,1\}, l \in [L]}, \{K_{j,i,l}^{(b)}\}_{j \in [n], b \in \{0,1\}, l \in [L]})$ . Compute  $\pi_i \leftarrow \text{prove}(crs, \phi_i, w_i)$ .
6. Broadcast  $(\pi_i, \text{msg}_i^1, \text{GC}_i, com_i)$ . Send  $\{o_{j,i,l}^{(b)}\}_{b \in \{0,1\}, l \in [L]}$  privately (over peer-to-peer channel) to  $P_j$  ( $j \in [n]$ ).

**Second Round (Asynchronous).** Each party  $P_i$  does the following:

1. If there exists  $j \in [n]$  such that  $\text{verify}(crs, \phi_j, \pi_j) \neq 1$ , send **abort** to all over peer-to-peer channel and output  $\perp$ .
2. For each  $j \in [n], l \in [L], b \in \{0,1\}$ : Compute  $K_{i,j,l}^{(b)} \leftarrow \text{open}(\text{pp}, com_{i,j,l}^{(b)}, o_{i,j,l}^{(b)})$  where  $o_{i,j,l}^{(b)}$  was received privately from  $P_j$  in Round 1 and  $com_{i,j,l}^{(b)} \in com_j$  was broadcast by  $P_j$  in Round 1. If there

exists any  $K_{i,j,l}^{(b)} = \perp$ , send **abort** to all over peer-to-peer channels and output  $\perp$ .

3. Let  $(\nu_1, \dots, \nu_L)$  denote the bits comprising  $(\text{msg}_1^1, \dots, \text{msg}_n^1)$ , where  $\text{msg}_j^1$  was broadcast by  $P_j$  in Round 1.

- For each  $(j \in [n])$  and  $l \in [L]$ , send  $o_{i,j,l} = o_{i,j,l}^{(\nu_l)}$  to each party  $P_k$  ( $k \in [n]$ ) over peer-to-peer channels.

**Output Computation.**  $P_i$  does the following after receiving second round messages from  $(n - t) > t$  parties (say whose indices constitute the set  $S_i$ )

1. If received **abort** from any party  $P_k$  where  $k \in S_i$ , output  $\perp$ .
2. Else, for each  $j \in [n], k \in S_i, l \in [L]$ , compute  $K_{k,j,l} \leftarrow \text{open}(\text{pp}, \text{com}_{k,j,l}^{(\nu_l)}, o_{k,j,l})$  where  $o_{k,j,l}$  was received privately from  $P_k$  in Round 2 and  $\text{com}_{k,j,l}^{(\nu_l)} \in \text{com}_j$  was broadcast by  $P_j$  in Round 1. If there exists any  $K_{k,j,l} = \perp$ , output  $\perp$ . Else, continue.
- For each  $j \in [n], l \in [L]$ , compute  $K_{j,l} \leftarrow \text{Shamir.reconstruct}(\{K_{k,j,l}\}_{k \in S_i})$ . If the reconstruction fails, output  $\perp$ .
3. Evaluate  $\text{msg}_j^2 \leftarrow \text{eval}(\text{GC}_j, (K_{j,1}, \dots, K_{j,L}))$ . If the evaluation fails, output  $\perp$ .
4. Output  $y \leftarrow \text{output}_i(x_i, r_i, \text{msg}_1^1, \dots, \text{msg}_n^1, \text{msg}_1^2, \dots, \text{msg}_n^2)$ .

---

*Two-round (where the first round is over synchronous broadcast and private peer-to-peer channels, and the second is over asynchronous peer-to-peer channel) secure computation in the CRS model with selective abort and corruption threshold  $t$ , where  $t < \frac{n}{2}$ .*

The intuition why this protocol is secure goes via a series of hybrids experiments that we sketch below. Let  $\mathcal{S}_{\text{bc}}$  the simulator of  $\Pi_{\text{bc}}$ .

- We start with a real execution; the simulator honestly plays the role of all honest parties, given their inputs.
- The simulator sends **abort** to the ideal functionality if one of the following holds: (a) the extractor **extract** of the NIZK fails w.r.t. any of the proofs sent by the adversary; (b) there exists a corrupt party who does not send a valid opening (consistent with the commitment that was broadcast) to any honest party. Otherwise, the simulator uses the simulator  $\mathcal{S}_{\text{bc}}$  to extract the inputs of adversary (i.e. impersonating for  $\mathcal{S}_{\text{bc}}$  an adversary of  $\Pi_{\text{bc}}$  and running internally  $\mathcal{A}$ ). The simulator sends the inputs to the ideal functionality (or abort based on  $\mathcal{S}_{\text{bc}}$ ) which will respond with the output  $y$ . Finally, note that if some parties abort in the real execution in the output phase, the simulator will send the corresponding indices to the ideal functionality since she computes the same checks in output phase as the honest parties do as per the protocol steps. This hybrid is indistinguishable from the previous one due to the security of  $\Pi_{\text{bc}}$ .

- The simulator uses the simulator of the NIZK to compute the proofs of the honest parties (that are sent in the first round). This hybrid is indistinguishable from the previous one due to the simulation-soundness property of the NIZK.
- All the commitments generated by the honest parties whose corresponding opening should be received by another honest party are switched to commitment computed in equivocal mode. In the second round the simulator equivocates the opening of these commitments (i.e. the one between honest parties) – in particular, she will send as opening the shares of labels that are computed in the first round (as it was done in the previous hybrid). This hybrid is indistinguishable from the previous one due to equivocal property of the commitment scheme.
- The simulator commits to random strings for the commitments that are generated by honest parties and whose respective openings are sent to adversary. Then in the second round the simulator runs `Shamir.simshare` to equivocate the shares of the honest parties, i.e. the shares of the honest parties are computed based on the labels of the garbled circuit (computed in the first round) and the random strings committed in the first round and delivered to the adversary. Note that in this case the commitment between honest parties are computed in equivocal mode, therefore the simulator simply equivocates to the share of the label of the garbled circuit computed as stated above. This hybrid is indistinguishable from the previous one due to the privacy of the secret sharing scheme.
- The simulator in round one, uses the simulator of the adaptive garbled circuit to compute the garble circuit of the honest parties. Then the simulator, in the end of round one, uses the second round messages of  $\Pi_{bc}$  as input for the simulator of the adaptive garbled scheme to obtain the correct input labels for the honest parties' garbled circuits. The simulator equivocates the shares of the labels of the honest parties as explained above. This hybrid is indistinguishable from the previous one due to the security of the adaptively secure garbling scheme.
- Finally, the simulator uses  $\mathcal{S}_{bc}$  to generate the messages of  $\Pi_{bc}$ , she does so interacting as a proxy between the ideal functionality and the adversary, and emulating an adversary of  $\Pi_{bc}$  to  $\mathcal{S}_{bc}$ . This hybrid is indistinguishable from the previous one due to the security of  $\Pi_{bc}$  and from the fact that the inputs of the adversary are extracted from the first round. This hybrid corresponds to the final simulator.