Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation*

Ran Cohen† Yehuda Lindell†

July 6, 2016

Abstract

In the setting of secure multiparty computation, a set of parties wish to compute a joint function of their private inputs. The computation should preserve security properties such as privacy, correctness, independence of inputs, fairness and guaranteed output delivery. In the case of no honest majority, fairness and guaranteed output delivery cannot always be obtained. Thus, protocols for secure multiparty computation are typically of two disparate types: protocols that assume an honest majority (and achieve all properties *including* fairness and guaranteed output delivery), and protocols that do not assume an honest majority (and achieve all properties *except for* fairness and guaranteed output delivery). In addition, in the two-party case, fairness and guaranteed output delivery are equivalent. As a result, the properties of fairness (which means that if corrupted parties receive output then so do the honest parties) and guaranteed output delivery (which means that corrupted parties cannot prevent the honest parties from receiving output in any case) have typically been considered to be the same.

In this paper, we initiate a study of the relation between fairness and guaranteed output delivery in secure multiparty computation. We show that in the multiparty setting these properties are distinct and proceed to study under what conditions fairness implies guaranteed output delivery (the opposite direction always holds). We also show the existence of non-trivial functions for which complete fairness is achievable (without an honest majority) but guaranteed output delivery is not, and the existence of non-trivial functions for which complete fairness and guaranteed output delivery are achievable. Our study sheds light on the role of broadcast in fairness and guaranteed output delivery, and shows that these properties should sometimes be considered separately.

Keywords: secure multiparty computation, theoretical foundations, identifiable abort, complete fairness, guaranteed output delivery, broadcast

*An extended abstract of this work appeared in [CL14]. This research was supported by THE ISRAEL SCIENCE FOUNDATION (grant No. 189/11). The first author was also supported by the Ministry of Science, Technology and Space and by the National Cyber Bureau of Israel.

†Department of Computer Science, Bar-Ilan University, Israel. email: cohenrb@cs.biu.ac.il, lindell@biu.ac.il

Contents

1 Introduction

1.1 Background

In the setting of secure multiparty computation, a set of mutually distrusting parties wish to jointly and securely compute a function of their inputs. This computation should be such that each party receives its correct output, and none of the parties learn anything beyond their prescribed output. In more detail, the most important security properties that we wish to capture are: privacy (no party should learn anything more than its prescribed output), correctness (each party is guaranteed that the output that it receives is correct), independence of inputs (the corrupted parties must choose their inputs independently of the honest parties' inputs), fairness[1] (corrupted parties should receive their output if and only if honest parties do), and guaranteed output delivery (corrupted parties should not be able to prevent honest parties from receiving their output). The standard definition today, [Can00, Gol04] formalizes the above requirements (and others) in the following general way. Consider an ideal world in which an external trusted party is willing to help the parties carry out their computation. An ideal computation takes place in this ideal world by having the parties simply send their inputs to the trusted party, who then computes the desired function and passes each party its prescribed output. The security of a real protocol is established by comparing the outcome of the protocol to the outcome of an ideal computation. Specifically, a real protocol that is run by the parties (without any trusted party) is secure, if an adversary controlling a coalition of corrupted parties can do no more harm in a real execution than in the above ideal execution.

The above informal description is "overly ideal" in the following sense. It is a known fact that unless an honest majority is assumed, it is impossible to obtain generic protocols for secure multiparty computation that guarantee output delivery and fairness [Cle86]. The definition is therefore typically *relaxed* when no honest majority is assumed. In particular, under certain circumstances, honest parties may not receive any output, and fairness is not always guaranteed. Recently, it was shown that it is actually possible to securely compute some (in fact, many) two-party functionalities fairly [GHKL08, Ash14]. In addition, it is possible to even compute some multiparty functionalities fairly, for any number of corrupted parties; in particular, the *majority* function may be securely computed fairly with three parties, and the *Boolean OR* function may be securely computed for any number of parties [GK09]. This has promoted interest in the question of fairness in the setting of no honest majority.

1.2 Fairness versus Guaranteed Output Delivery

The two notions of fairness and of guaranteed output delivery are quite similar and are often interchanged. However, there is a fundamental difference between them. If a protocol guarantees output delivery, then the parties always obtain output and cannot abort. In contrast, if a protocol is fair, then it is only guaranteed that *if* one party receives output then all parties receive output. Thus, it is possible that all parties abort. In order to emphasize the difference between the notions, we note that every protocol that provides guaranteed output delivery can be transformed into a protocol that provides fairness but *not* guaranteed output delivery, as follows. At the beginning every party broadcasts OK; if one of the parties did not send OK then all the parties output $\perp$; otherwise

[1] Throughout this paper, whenever we say "fair" we mean "completely fair", and so if any party learns anything then all parties receive their entire output. This is in contrast to notions of partial fairness that have been studied in the past.

the parties execute the original protocol (that ensures guaranteed output delivery). Clearly every party can cause the protocol to abort. However, it can only do so before any information has been obtained. Thus, the resulting protocol is fair, but does not guarantee output delivery.

It is immediate to see that guaranteed output delivery implies fairness, since if all parties must receive output then it is not possible for the corrupted parties to receive output while the honest do not. However, the opposite direction is not clear. In the two-party case, guaranteed output delivery is indeed implied by fairness since upon receiving abort, the honest party can just compute the function on its own input and a default input for the other party. However, when there are many parties involved, it is not possible to replace inputs with default inputs since the honest parties do not necessarily know who is corrupted (and security mandates that honest parties' inputs cannot be changed; otherwise, this could be disastrous in an election-type setting). This leads us to the following fundamental questions, which until now have not been considered at all (indeed, fairness and guaranteed output delivery are typically used synonymously):

> *Does fairness imply guaranteed output delivery? Do there exist functionalities that can be securely computed with fairness but not with guaranteed output delivery? Are there conditions on the function/network model for which fairness implies guaranteed output delivery?*

The starting point of our work is the observation that the *broadcast functionality* does actually separate guaranteed output delivery and fairness. Specifically, let n denote the overall number of parties, and let t denote an upper bound on the number of corrupted parties. Then, it is well known that secure broadcast can be achieved if and only if $t < n/3$ [PSL80, LSP82]. However, it is also possible to achieve *weak* broadcast (which means that either all parties abort and no one receives output, or all parties receive and agree upon the broadcasted value) for any $t < n$ [FGH+02]. In our terms, this is a secure computation of the broadcast functionality with *fairness* but *no guaranteed output delivery* . Thus, we see that for $t \geq n/3$ there exist functionalities that can be securely computed with fairness but not with guaranteed output delivery (the fact that broadcast cannot be securely computed with guaranteed output delivery for $t \geq n/3$ follows directly from the bounds on Byzantine Generals [PSL80, LSP82]). Although broadcast does provide a separation, it is an atypical function. Specifically, there is no notion of privacy, and the functionality can be computed information theoretically for any $t < n$ given a secure setup phase [PW92]. Thus, broadcast is a trivial functionality.[2] This leaves the question of whether fairness and guaranteed output delivery are distinct still holds for more "standard" secure computation tasks.

It is well known that for $t < n/2$ any multiparty functionality can be securely computed with guaranteed output delivery given a broadcast channel [GMW87, RB89]. Fitzi et al. [FGMvR02] used weak broadcast in the protocols of [GMW87, RB89] and showed that *any* functionality can be securely computed with fairness for $t < n/2$. This leaves open the question as to whether there exist functionalities (apart from broadcast) that *cannot* be securely computed with guaranteed output delivery for $n/3 \leq t < n/2$.

Gordon and Katz [GK09] showed that the three-party majority function and multiparty Boolean OR function can be securely computed with guaranteed output delivery for *any* number of corrupted parties (in particular, with an honest minority). However, the constructions of [GK09] use a broadcast channel. This leads us to the following questions for the range of $t \geq n/3$:

[2] We stress that "trivial" does not mean easy to achieve or uninteresting. Rather, it means that cryptographic hardness is not needed to achieve it in the setting of no honest majority [Kil91].

1. Can the three-party majority function and multiparty Boolean OR function be securely computed with guaranteed output delivery without a broadcast channel?

2. Can the three-party majority function and multiparty Boolean OR function be securely computed with fairness without a broadcast channel?

3. Does the existence of broadcast make a difference with respect to fairness and/or guaranteed output delivery *in general*?

We remark that conceptually guaranteed output delivery is a stronger notion of security and that it is what is required in some applications. Consider the application of "mental poker"; if guaranteed output delivery is not achieved, then a corrupted party can cause the execution to abort in case it is dealt a bad hand. This is clearly undesirable.

1.3 Our Results

Separating fairness and guaranteed output delivery. We show that the three-party majority function, which can be securely computed with fairness [GK09], *cannot* be securely computed with guaranteed output delivery. Thus, there exist non-trivial functionalities (i.e., functionalities that cannot be securely computed in the information-theoretic setting without an honest majority) for which fairness can be achieved but guaranteed output delivery cannot. Technically, we show this by proving that the three-party majority function can be used to achieve broadcast, implying that it cannot be securely computed with guaranteed output delivery.

Theorem 1.1 (informal)**.** *Consider a model without a broadcast channel and consider any $t \geq n/3$. Then, there exist non-trivial functionalities f (e.g., the majority function) such that f can be securely computed with fairness but f cannot be securely computed with guaranteed output delivery.*

This proves that fairness and guaranteed output delivery are distinct, at least in a model without a broadcast channel.

Feasibility of guaranteed output delivery without broadcast. The protocols of [GK09] for majority and Boolean OR both use a broadcast channel to achieve guaranteed output delivery. As we have seen in Theorem 1.1 this is essential for achieving their result for the majority function. However, is this also the case for the Boolean OR function? In general, do there exist non-trivial functionalities for which guaranteed output delivery is achievable without a broadcast channel and for any number of corrupted parties?

Theorem 1.2 (informal)**.** *Consider a model without a broadcast channel and consider any number of corruptions. Then, there exist non-trivial functionalities f (e.g., the Boolean OR function) such that f can be securely computed with guaranteed output delivery.*

On the role of broadcast. We show that the existence or non-existence of broadcast is meaningless with respect to fairness, but of great significance with respect to guaranteed output delivery. Specifically, we show the following:

Theorem 1.3 (informal)**.** *Let f be a multiparty functionality. Then:*

1. *There exists a protocol for securely computing f with fairness with a broadcast channel* if and only if *there exists a protocol for securely computing f with fairness without a broadcast channel.*

2. *If there exists a protocol for securely computing f with fairness (with or without a broadcast channel), then there exists a protocol for securely computing f with guaranteed output delivery with a broadcast channel.*

Thus, fairness and guaranteed output delivery are *equivalent* in a model with a broadcast channel, and *distinct* without a broadcast channel. In contrast, by Theorem 1.1 we already know that without broadcast it does not hold that fairness implies guaranteed output delivery (otherwise, the separation in Theorem 1.1 would not be possible). We also show that under black-box reductions, fairness *never* helps achieve guaranteed output delivery. That is:

Theorem 1.4 (informal)**.** *Let f be a multiparty functionality and consider a hybrid model where a trusted party computes f fairly for the parties (i.e., either all parties receive output or none do). Then, there exists a protocol for securely computing f with guaranteed output delivery in this hybrid model if and only if there exists a protocol for securely computing f with guaranteed output delivery in the real model with no trusted party.*

Intuitively, Theorem 1.4 follows from the fact that an adversary can always cause the result of calls to f to be abort in which case they are of no help. This does not contradict item (2) of Theorem 1.3 since given a broadcast channel and non-black-box access to the protocol that computes f with fairness, it is possible to apply a variant of the GMW compiler [GMW87] and detect which party cheated and caused the abort to occur.

Conditions under which fairness implies guaranteed output delivery. We have already seen that fairness implies guaranteed output delivery given broadcast. We also consider additional scenarios in which fairness implies guaranteed output delivery. We prove that if a functionality can be securely computed with fairness and *identifiable abort* (meaning that the identity of the cheating party is detected) then the functionality can be securely computed with guaranteed output delivery. Finally, we show that in the fail-stop model (where the only thing an adversary can do is instruct a corrupted party to halt prematurely), fairness is always equivalent to guaranteed output delivery. This follows from the fact that broadcast is trivial in the fail-stop model.

Identifiable abort and broadcast. In the model of identifiable abort, the identity of the cheating party is revealed to the honest parties. This definition was explicitly used by Aumann and Lindell [AL10], who remarked that it is met by most protocols (e.g., [GMW87]), but not all (e.g., [GL05]). This model has the advantage that a cheating adversary who runs a "denial of service" attack and causes the protocol to abort cannot go undetected. Thus, it cannot repeatedly prevent the parties from obtaining output. An interesting corollary that comes out of our work—albeit not related to fairness and guaranteed output delivery—is that security with identifiable abort *cannot* be achieved in general for $t \geq n/3$ without broadcast. This follows from the fact that if identifiable abort can be achieved in general (even without fairness), then it is possible to achieve broadcast. Thus, we conclude:

Corollary 1.5 (informal)**.** *Consider a model without a broadcast channel and consider any $t \geq n/3$. Then, there exist functionalities f that cannot be securely computed with identifiable abort.*

Summary of feasibility. Table 1 below summarizes the state of affairs regarding feasibility for secure computation with fairness and guaranteed output delivery, for different ranges regarding the number of corrupted parties.

<table>
<tr><th>Number of Corrupted</th><th>With Broadcast</th><th>Without Broadcast</th></tr>
<tr><td>$t < n/3$</td><td colspan="2">All f can be securely computed
with guaranteed output delivery</td></tr>
<tr><td>$n/3 \leq t < n/2$</td><td>All f can be computed with
guaranteed output delivery</td><td rowspan="2">f_{or} can be computed with
guaranteed output delivery
f_{maj} cannot be computed with
guaranteed output delivery</td></tr>
<tr><td>$t \geq n/2$</td><td>Fairness implies
guaranteed output delivery</td></tr>
<tr><td>$t < n$</td><td colspan="2">f can be securely computed fairly with broadcast iff
f can be securely computed fairly without broadcast</td></tr>
</table>

Table 1: Feasibility of fairness and guaranteed output delivery

Preliminaries. Full definitions can be found in Appendix A. We consider a number of different ideal models: security with guaranteed output delivery, with fairness, with abort, with identifiable abort (meaning that in the case of abort one of the corrupted parties is identified by the honest parties), and fairness with identifiable abort. The ideal models for these models are respectively denoted $\text{IDEAL}^{\mathsf{g.d.}}$, $\text{IDEAL}^{\mathsf{fair}}$, $\text{IDEAL}^{\mathsf{abort}}$, $\text{IDEAL}^{\mathsf{id\text{-}abort}}$, $\text{IDEAL}^{\mathsf{id\text{-}fair}}$. We also consider hybrid model protocols where the parties send regular messages to each other, and also have access to a trusted party who computes some function f for them. The trusted party may compute according to any of the specified ideal model. Letting $\mathsf{type} \in \{\mathsf{g.d.}, \mathsf{fair}, \mathsf{abort}, \mathsf{id\text{-}abort}, \mathsf{id\text{-}fair}\}$, we call this the (f, type)-hybrid model, and denote it $\text{HYBRID}^{f,\mathsf{type}}$. The security parameter is denoted by κ, and the set of corrupted parties by $\mathcal{I}$. Unless stated otherwise, all adversaries considered are *malicious*.

2 Separating Fairness from Guaranteed Output Delivery

In this section we prove Theorem 1.1. As we have mentioned in the Introduction, it is known that secure broadcast can be t-securely computed with guaranteed output delivery if and only if $t < n/3$. In addition, secure broadcast can be computed with fairness, for any $t \leq n$, using the protocol of [FGH+02]. Thus, broadcast already constitutes a separation of fairness from guaranteed output delivery; however, since broadcast can be information-theoretically computed (and is trivial in the technical sense; see Footnote 2), we ask whether or not such a separation also exists for more standard secure computation tasks.

In order to show a separation, we need to take a function for which fairness in the multiparty setting is feasible. Very few such functions are known, and the focus of this paper is not the construction of new protocols. Fortunately, Gordon and Katz [GK09] showed that the three-party majority function can be securely computed with fairness. (In [GK09] a broadcast channel is used. However, as we show in Section 4.1, this implies the result also without a broadcast channel.)

We stress that the three-party majority function is not trivial, and in fact the ability to securely compute it with any number of corruptions implies the existence of oblivious transfer (this is shown by reducing the two-party greater-than functionality to it and applying [Kil91]).

We show that the three-party majority function f_{maj} *cannot* be securely computed with guaranteed output delivery and any number of corrupted parties in the point-to-point network model by showing that it actually implies broadcast. The key observation is that there exists an input $(1,1,1)$ for which the output of f_{maj} will be 1, even if a single corrupted party changes its input to 0. Similarly, there exists an input $(0,0,0)$ for which the output of f_{maj} will be 0, even if a single corrupt party changes its input to 1. Using this property, we show that if f_{maj} can be computed with guaranteed output delivery, then there exists a broadcast protocol for three parties that is secure against a single corruption. Given an input bit β, the sender sends β to each other party, and all parties compute f_{maj} on the input they received. This works since a corrupted dealer cannot make two honest parties output inconsistent values, since f_{maj} provides the same output to all parties. Likewise, if there is one corrupted receiver, then it cannot change the majority value (as described above). Finally, if there are two corrupted receivers, then it makes no difference what they output anyway.

Theorem 2.1. *Let t be a parameter and let $f_{\mathsf{maj}} : \{0,1\}^3 \to \{0,1\}^3$ be the majority functionality for three parties $f_{\mathsf{maj}}(x_1,x_2,x_3) = (y,y,y)$ where $y = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \wedge (x_2 \wedge x_3)$. If f_{maj} can be t-securely computed with guaranteed output delivery in a point-to-point network, then there exists a protocol that t-securely computes the three-party broadcast functionality.*

Proof: We construct a protocol π for securely computing the three-party broadcast functionality $f_{\mathsf{bc}}(x,\lambda,\lambda) = (x,x,x)$ in the $(f_{\mathsf{maj}}, \mathsf{g.d.})$-hybrid model (i.e., in a hybrid model where a trusted party computes the f_{maj} functionality with guaranteed output delivery). Protocol π works as follows:

1. The sender P_1 with input $x \in \{0,1\}$ sends x to P_2 and P_3.
2. Party P_1 sends x to the trusted party computing f_{maj}. Each party P_i ($i \in \{2,3\}$) sends the value it received from P_1 to f_{maj}.
3. Party P_1 always outputs x. The parties P_2 and P_3 output whatever they receive from the trusted party computing f_{maj}.

Let $\mathcal{A}$ be an adversary attacking the execution of π in the $(f_{\mathsf{maj}}, \mathsf{g.d.})$-hybrid model; we construct an ideal-model adversary $\mathcal{S}$ in the ideal model for f_{bc} with guaranteed output delivery. $\mathcal{S}$ invokes $\mathcal{A}$ and simulates the interaction of $\mathcal{A}$ with the honest parties and with the trusted party computing f_{maj}. $\mathcal{S}$ proceeds based on the following corruption cases:

- *P_1 alone is corrupted:* $\mathcal{S}$ receives from $\mathcal{A}$ the values $x_2, x_3 \in \{0,1\}$ that it sends to parties P_2 and P_3, respectively. Next, $\mathcal{S}$ receives the value $x_1 \in \{0,1\}$ that $\mathcal{A}$ sends to f_{maj}. $\mathcal{S}$ computes $x = f_{\mathsf{maj}}(x_1,x_2,x_3)$ and sends x to the trusted party computing f_{bc}. $\mathcal{S}$ simulates $\mathcal{A}$ receiving x back from f_{maj}, and outputs whatever $\mathcal{A}$ outputs.
- *P_1 and one of P_2 or P_3 are corrupted:* the simulation is the same as in the previous case except that if P_2 is corrupted then the value x_2 is taken from what $\mathcal{A}$ sends in the name of P_2 to f_{maj} (and not the value that $\mathcal{A}$ sends first to P_2); likewise for P_3. Everything else is the same.

- P_1 *is honest:* $\mathcal{S}$ sends an empty input λ to the trusted party for every corrupted party, and receives back some $x \in \{0,1\}$. Next, $\mathcal{S}$ simulates P_1 sending x to both P_2 and P_3. If both P_2 and P_3 are corrupted, then $\mathcal{S}$ obtains from $\mathcal{A}$ the values x_2 and x_3 that they send to f_{maj}, computes $x' = f_{\mathsf{maj}}(x, x_2, x_3)$ and simulates the trusted party sending x' back to all parties. If only one of P_2 and P_3 are corrupted, then $\mathcal{S}$ simulates the trusted party sending x back to all parties. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

The fact that the simulation is good is straightforward. If P_1 is corrupted, then only consistency is important, and $\mathcal{S}$ ensures that the value sent to f_{bc} is the one that the honest party/parties would output. If P_1 is not corrupted, and both P_2 and P_3 are corrupted, then P_1 always outputs the correct x as required, and the outputs of P_2 and P_3 are not important. Finally, if P_1 and P_2 are corrupted, then $\mathcal{S}$ sends f_{bc} the value that P_3 would output in the real protocol as required; likewise for P_1 and P_3 corrupted. ■

Theorem 2.1 implies that f_{maj} cannot be securely computed with guaranteed output delivery for any $t < 3$ in a point-to-point network; this follows immediately from the fact that the broadcast functionality can be securely computed if and only if $t < n/3$. Furthermore, by [GK09], f_{maj} *can* be securely computed fairly given oblivious transfer (and as shown in Section 4.1 this also holds in a point-to-point network). Thus, we have:

Corollary 2.2. *Assume that oblivious transfer exists. Then, there exist non-trivial functionalities f such that f can be t-securely computed with fairness but cannot be t-securely computed with guaranteed output delivery, in a point-to-point network and with $t \geq n/3$.*

Three-party functionalities that imply broadcast. It is possible to generalize the property that we used to show that f_{maj} implies broadcast. Specifically, consider a functionality f with the property that there exist inputs (x_1, x_2, x_3) and (x'_1, x'_2, x'_3) such that $f(x_1, x_2, x_3) = 0$ and $f(x'_1, x'_2, x'_3) = 1$, and such that if either of x_2 or x_3 (resp., x'_2 or x'_3) are changed arbitrarily, then the output of f remains the same. Then, this function can be used to achieve broadcast. We describe the required property formally inside the proof of the theorem below. We show that out of the 256 functions over 3-bit inputs, there are 110 of them with this property. It follows that none of these can be securely computed with guaranteed output delivery in the presence of one or two corrupted parties. We prove the following:

Theorem 2.3. *There are* 110 *functions from the family of all three-party Boolean functions*

$$\{f \colon \{0,1\} \times \{0,1\} \times \{0,1\} \to \{0,1\}\}$$

that cannot be securely computed with guaranteed output delivery in a point-to-point network with $t = 1$ or $t = 2$.

Proof. We provide a combinatorial proof of the theorem, by counting how many functions have the property that arbitrarily changing one of the inputs does not affect the output, and there are inputs that yield output 0 and inputs that yield output 1. As we have seen in the proof of Theorem 2.1, it is possible to securely realize the broadcast functionality given a protocol that securely computes any such functionality with guaranteed output delivery.

We prove that there are 110 functions $f \colon \{0,1\}^3 \to \{0,1\}$ in the union of the following sets F_1, F_2, F_3:

1. Let F_1 be the set of all functions for which there exist $(a,b,c),(a',b',c') \in \{0,1\}^3$ such that $f(a,b,\cdot) = f(a,\cdot,c) = 1$ and $f(a',b',\cdot) = f(a',\cdot,c') = 0$.
2. Let F_2 be the set of all functions for which there exist $(a,b,c),(a',b',c') \in \{0,1\}^3$ such that $f(a,b,\cdot) = f(\cdot,b,c) = 1$ and $f(a',b',\cdot) = f(\cdot,b',c') = 0$.
3. Let F_3 be the set of all functions for which there exist $(a,b,c),(a',b',c') \in \{0,1\}^3$ such that $f(\cdot,b,c) = f(a,\cdot,c) = 1$ and $f(\cdot,b',c') = f(a',\cdot,c') = 0$.

Observe that any function in one of these sets can be used to achieve broadcast, as described above. Based on the inclusion-exclusion principle and using Lemma 2.5 proven below, it follows that:

$$|F_1 \cup F_2 \cup F_3| = 3 \cdot 50 - 3 \cdot 16 + 8 = 110,$$

as required. We first prove the following lemma:

Lemma 2.4. *If $f \in F_1$, then $a \neq a'$, if $f \in F_2$ then $b \neq b'$ and if $f \in F_3$ then $c \neq c'$.*

Proof. Let $f \in F_1$ and let $a, a', b, b', c, c' \in \{0,1\}$ be inputs fulfilling the condition for set F_1. Assume by contradiction that $a = a'$. Thus,

$$f(a,b,c) = f(a,\bar{b},c) = f(a,b,\bar{c}) = 1 \quad \text{and} \quad f(a,b',c') = f(a,\bar{b}',c') = f(a,b',\bar{c}') = 0.$$

If $b = b'$ then $f(a,b,c') = f(a,b',c') = 0$. However, $f(a,b,c) = f(a,b,\bar{c}) = 1$ and so $f(a,b,c') = 1$ for any c', in contradiction. Thus $b \neq b'$. Similarly, $c \neq c'$. Therefore, $b' = \bar{b}$ and $c' = \bar{c}$ and by the condition, $f(a,b,c) = 1$ and $f(a,\bar{b},\bar{c}) = 0$.

Consider $f(a,\bar{b},c)$. From the condition, $f(a,b,c) = f(a,\bar{b},c) = 1$. However, changing the c coordinate to $\bar{c}$ gives us $f(a,\bar{b},\bar{c})$ which by the condition equals 0 (because $b' = \bar{b}$ and $c' = \bar{c}$). We therefore derive a contradiction, and so conclude that $a' = \bar{a}$. ∎

It remains to prove the following lemma, to derive the theorem.

Lemma 2.5.

1. $|F_1| = |F_2| = |F_3| = 50$.
2. $|F_1 \cap F_2| = |F_1 \cap F_3| = |F_2 \cap F_3| = 16$.
3. $|F_1 \cap F_2 \cap F_3| = 8$.

Proof. Let $f\colon \{0,1\}^3 \to \{0,1\}$ be a function, and consider the representation of f using a binary string $(\beta_0\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7)$ as shown in Table 2:

0	0	0	β_0
0	0	1	β_1
0	1	0	β_2
0	1	1	β_3
1	0	0	β_4
1	0	1	β_5
1	1	0	β_6
1	1	1	β_7

Table 2: Representation of a Boolean function $\{0,1\}^3 \to \{0,1\}$

1. Assume $f \in F_1$ (the proof for F_2, F_3 is similar). The first quadruple $(\beta_0\beta_1\beta_2\beta_3)$ corresponds to $a = 0$ and the second quadruple $(\beta_4\beta_5\beta_6\beta_7)$ corresponds to $a = 1$. There exists b, c such that $f(a, b, c) = f(a, \bar{b}, c) = f(a, b, \bar{c})$ and b', c' such that $f(\bar{a}, b', c') = f(\bar{a}, \bar{b}', c') = f(\bar{a}, b', \bar{c}')$, in addition, $f(a, b, c) \neq f(\bar{a}, b', c')$. Therefore, in each such quadruple there must be a triplet of 3 identical bits, and the two triplets have opposite values.

   Denote $\beta = f(a, b, c)$, there are 5 options for $(\beta_0\beta_1\beta_2\beta_3)$ in which 3 of the bits equal β:

   $$(\beta\beta\beta\beta), (\beta\beta\beta\bar{\beta}), (\beta\beta\bar{\beta}\beta), (\beta\bar{\beta}\beta\beta), (\bar{\beta}\beta\beta\beta).$$

   For each such option, there are 5 options for $(\beta_4\beta_5\beta_6\beta_7)$ in which 3 of the bits equal $\bar{\beta}$:

   $$(\bar{\beta}\bar{\beta}\bar{\beta}\bar{\beta}), (\bar{\beta}\bar{\beta}\bar{\beta}\beta), (\bar{\beta}\bar{\beta}\beta\bar{\beta}), (\bar{\beta}\beta\bar{\beta}\bar{\beta}), (\beta\bar{\beta}\bar{\beta}\bar{\beta}).$$

   There are 2 options for the value of β, so in total $|F_1| = 2 \cdot 5 \cdot 5 = 50$.

2. Assume $f \in F_1 \cap F_2$ (the proof for $F_1 \cap F_3, F_2 \cap F_3$ is similar). In this case $a' = \bar{a}$ and $b' = \bar{b}$ and the constraints are

   $$f(a, b, c) = f(\bar{a}, b, c) = f(a, \bar{b}, c) = f(a, b, \bar{c}) \neq f(\bar{a}, \bar{b}, c') = f(a, \bar{b}, c') = f(\bar{a}, b, c') = f(\bar{a}, \bar{b}, \bar{c}').$$

   Therefore, the string is balanced (there are 4 zeros and 4 ones), where 3 of the bits $(\beta_0\beta_1\beta_2\beta_3)$ are equal to β and one to $\bar{\beta}$, and 3 of the bits $(\beta_4\beta_5\beta_6\beta_7)$ are equal to $\bar{\beta}$ and one to β.

   There are 4 options to select 3 bits in $(\beta_0\beta_1\beta_2\beta_3)$, and 2 options to select one bit in $(\beta_4\beta_5\beta_6\beta_7)$. These two options correspond either to $(\bar{a}, b, c)$ or $(\bar{a}, \bar{b}, \bar{c})$. Hence, $|F_1 \cap F_2| = 2 \cdot 4 \cdot 2 = 16$.

3. Assume $f \in F_1 \cap F_2 \cap F_3$. In this case $a' = \bar{a}$, $b' = \bar{b}$ and $c' = \bar{c}$ and the constraints are

   $$f(a, b, c) = f(\bar{a}, b, c) = f(a, \bar{b}, c) = f(a, b, \bar{c}) \neq f(\bar{a}, \bar{b}, \bar{c}) = f(a, \bar{b}, \bar{c}) = f(\bar{a}, b, \bar{c}) = f(\bar{a}, \bar{b}, c).$$

   Therefore, the string is of the form $(\beta_0\beta_1\beta_2\beta_3\bar{\beta}_0\bar{\beta}_1\bar{\beta}_2\bar{\beta}_3)$, where 3 of the bits $(\beta_0\beta_1\beta_2\beta_3)$ are equal to β and one to $\bar{\beta}$.

   There are 4 options to select 3 bits in $(\beta_0\beta_1\beta_2\beta_3)$, and setting them to the same value determines the rest of the string. Hence, $|F_1 \cap F_2 \cap F_3| = 2 \cdot 4 = 8$.

■

This completes the proof of Theorem 2.3. ■

As we have mentioned in the Introduction, in the case that $t = 1$ (i.e., when there is an honest majority), all functions can be securely computed with fairness in a point-to-point network. Thus, we have that all 110 functions of Theorem 2.3 constitute a *separation* of fairness from guaranteed output delivery. That is, in the case of $n/3 \leq t < n/2$, we have that *many functions* can be securely computed with fairness but not with guaranteed output delivery. In addition, 8 out of these 110 functions reduce to three-party majority and so can be computed fairly for any $t \leq n$. Thus, these 8 functions form a separation for the range of $t \geq n/2$.

3 Fairness Implies Guaranteed Output Delivery for Default-Output Functionalities

In this section we prove Theorem 1.2. In fact, we prove a stronger theorem, stating that fairness implies guaranteed output delivery for functions with the property that there exists a "default value" such that any single party can fully determine the output to that value. For example, the multiparty Boolean AND and OR functionalities both have this property (for the AND functionality any party can always force the output to be 0, and for the OR functionality any party can always force the output to be 1). We call such a function a default-output functionality. Intuitively, such a function can be securely computed with guaranteed output delivery if it can be securely computed fairly, since the parties can first try to compute it fairly. If they succeed, then they are done. Otherwise, they all received abort and can just output their respective default-output value for the functionality. This can be simulated since any single corrupted party in the ideal model can choose an input that results in the default output value.

Definition 3.1. *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. f is called a* default-output functionality *with default output $(\tilde{y}_1, \ldots, \tilde{y}_n)$, if for every $i \in \{1, \ldots, n\}$ there exists a special input $\tilde{x}_i$ such that for every x_j with $j \neq i$ it holds that $f(x_1, \ldots, \tilde{x}_i, \ldots, x_n) = (\tilde{y}_1, \ldots, \tilde{y}_n)$.*

Observe that $(0, \ldots, 0)$ is a default output for the Boolean AND function, and $(1, \ldots, 1)$ is a default output for the Boolean OR function. We now prove that if a functionality f has a default output value, then the existence of a fair protocol for f implies the existence of a protocol with guaranteed output delivery for f.

Theorem 3.2. *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a default-output functionality and let $t < n$. If f can be t-securely computed with fairness (with or without a broadcast channel), then f can be t-securely computed with guaranteed output delivery, in a point-to-point network.*

Proof. Let f be as in the theorem statement, and let the default output be $(\tilde{y}_1, \ldots, \tilde{y}_n)$. Assume that f can be securely computed with fairness with or without a broadcast channel. By Theorem 4.1, f can be securely computed with fairness without a broadcast channel. We now construct a protocol π that securely computes f with guaranteed output delivery in the (f, fair)-hybrid model:

1. Each P_i sends its input x_i to the trusted party computing f.
2. Denote by y_i the value received by P_i from the trusted party.
3. If $y_i \neq \perp$, P_i outputs y_i, otherwise P_i outputs $\tilde{y}_i$.

Let $\mathcal{A}$ be an adversary attacking the execution of π in the (f, fair)-hybrid model. We construct an ideal model adversary $\mathcal{S}$ in the ideal model with guaranteed output delivery. Let $\mathcal{I}$ be the set of corrupted parties, let $i \in \mathcal{I}$ be one of the corrupted parties (if no parties are corrupted then there is nothing to simulate), and let $\tilde{x}_i$ be the input guaranteed to exist by Definition 3.1. Then, $\mathcal{S}$ invokes $\mathcal{A}$ and simulates the interaction of $\mathcal{A}$ with the trusted party computing f (note that there is no interaction between $\mathcal{A}$ and honest parties). $\mathcal{S}$ receives the inputs that $\mathcal{A}$ sends to f. If any of the inputs equal abort then $\mathcal{S}$ sends $\tilde{x}_i$ as P_i's input to its own trusted party computing f (with guaranteed output delivery), and arbitrary inputs for the other parties. Then, $\mathcal{S}$ simulates the corrupted parties receiving $\perp$ as output from the trusted party in π, and outputs whatever $\mathcal{A}$

outputs. Else, if none of the inputs equal abort, then $\mathcal{S}$ sends its trusted party the inputs that $\mathcal{A}$ sent. $\mathcal{S}$ then receives the outputs of the corrupted parties from its trusted party, and internally sends these to $\mathcal{A}$ as the corrupted parties' outputs from the trusted party computing f in π. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{A}$ sends abort, then in the real execution every honest party P_j outputs $\tilde{y}_j$. However, since $\mathcal{S}$ sends the input $\tilde{x}_i$ to the trusted party computing f, by Definition 3.1 we have that the output of every honest party P_j in the ideal execution is also $\tilde{y}_j$. Furthermore, if $\mathcal{A}$ does not send abort, then $\mathcal{S}$ just uses exactly the same inputs that $\mathcal{A}$ sent. It is clear that the view of $\mathcal{A}$ is identical in the execution of π and the simulation with $\mathcal{S}$. We therefore conclude that π t-securely computes f with guaranteed output delivery, as required. ■

We have proven that fairness implies guaranteed output delivery for default-output functionalities; it remains to show the existence of fair protocols for some default-output functionalities. Fortunately, this was already proven in [GK09]. The only difference is that [GK09] uses a broadcast channel. Noting that the multiparty Boolean OR functionality is non-trivial (in the sense of Footnote 2), and that it has default output $(1, \ldots, 1)$ as mentioned above, we have the following corollary.

Corollary 3.3. *Assume that oblivious transfer exists. Then, there exist non-trivial functionalities f that can be t-securely computed with guaranteed output delivery in a point-to-point network, for any $t < n$.*

Feasibility of guaranteed output delivery. In Theorem 3.4, we prove that 16 non-trivial functionalities can be securely computed with guaranteed output delivery in a point-to-point network (by showing that they are default-output functionalities). Thus, guaranteed output delivery can be achieved for a significant number of functions.

Theorem 3.4. *Assume that oblivious transfer exists. There are* 16 *non-trivial functions from the family of all three-party Boolean functions $\{f\colon \{0,1\}\times\{0,1\}\times\{0,1\} \to \{0,1\}\}$ that can be securely computed with guaranteed output delivery in a point-to-point network for any number of corrupted parties.*

Proof. When represented using its truth table as a binary string (see Table 2), the three-party Boolean OR function is (01111111), similarly, the Boolean AND function is (00000001). Every function $(\beta_0\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7)$ such that there exists i for which $\beta_i = \beta$ and for every $j \neq i$ $\beta_j = \bar{\beta}$ can be reduced to computing Boolean OR. Since there are 8 ways to choose i and 2 ways to choose β, we conclude that there are 16 such functions. ■

4 The Role of Broadcast

In this section, we prove Theorem 1.3, and show that a functionality can be securely computed fairly with broadcast if and only if it can be securely computed fairly without broadcast. In addition, we show that if a functionality can be securely computed with fairness, then with a broadcast channel it can be securely computed with guaranteed output delivery.

4.1 Fairness is Invariant to Broadcast

Gordon and Katz construct two fair multiparty protocols in [GK09], both of them require a broadcast channel. In this section we show that fairness holds for both even without a broadcast channel. More generally, fairness can be achieved with a broadcast channel if and only if it can be achieved without a broadcast channel.

It is immediate that fairness without broadcast implies fairness with broadcast. The other direction follows by using the protocol of Fitzi et al. [FGH+02] for detectable broadcast. In the first stage, the parties execute a protocol that establishes a public-key infrastructure. This protocol is independent of the parties' inputs and is computed with abort. If the adversary aborts during this phase, it learns nothing about the output and fairness is retained. If the adversary does not abort, the parties can use the public-key infrastructure and execute multiple (sequential) instances of authenticated broadcast, and so can run the original protocol with broadcast that is fair.

One subtlety arises since the composition theorem replaces every ideal call to the broadcast functionality with a protocol computing broadcast. However, in this case, each authenticated broadcast protocol relies on the same public-key infrastructure that is generated using a protocol with abort. We therefore define a reactive ideal functionality which allows abort only in the first "setup" call. If no abort was sent in this call, then the functionality provides a fully secure broadcast (with guaranteed output delivery) from there on. The protocol of [FGH+02] securely computes this functionality with guaranteed output delivery, and thus constitutes a sound replacement of the broadcast channel (unless an abort took place).

Theorem 4.1. *Let f be an n-party functionality and let $t \leq n$. Then, assuming the existence of one-way functions, f can be t-securely computed with fairness assuming a broadcast channel if and only if f can be t-securely computed with fairness in a point-to-point network.*

Proof Sketch: If f can be t-securely computed with fairness in a point-to-point network, then it can be t-securely computed with fairness with a broadcast channel by just having parties broadcast messages and stating who the intended recipient is. (Recall that in the point-to-point network we assume authenticated but not private channels.)

Next, assume that f can be t-securely computed with fairness assuming a broadcast channel. We now show that it can be t-securely computed with fairness in a point-to-point network. We define the reactive functionality for *conditional broadcast* f_{condbc}. In the first call to f_{condbc}, the functionality computes the AND function, i.e., each party has an input bit b_i and the functionality returns $b = b_1 \wedge \ldots \wedge b_n$ to each party. In addition, the functionality stores the bit b as its internal state for all future calls. In all future calls to f_{condbc}, if $b = 1$ it behaves exactly like f_{bc}, whereas if $b = 0$ it returns $\perp$ to all the parties in the first call and halts. By inspection, it is immediate that the protocol of [FGH+02] securely computes f_{condbc} with guaranteed output delivery, for any $t \leq n$ in a point-to-point network.

Let π be the protocol that t-securely computes f assuming a broadcast channel; stated differently, π t-securely computes f in the $(f_{\mathsf{bc}}, \mathsf{g.d.})$-hybrid model. We construct a protocol π' for t-securely computing f in the $(f_{\mathsf{condbc}}, \mathsf{fair})$-hybrid model. π' begins by all parties sending the bit 1 to f_{condbc} and receiving back output. If a party receives back $b = 0$, it aborts and outputs $\perp$. Else, it runs π with the only difference that all broadcast messages are sent to f_{condbc} instead of to f_{bc}. Since f_{condbc} behaves exactly like f_{bc} as long $b = 1$ is returned from the first call, we have that in this case the output of π and π' is identical. Furthermore, π' is easily simulated by first invoking the adversary $\mathcal{A}'$ for π' and obtaining the corrupted parties' inputs to f_{condbc} in the first call. If

any 0 bit is sent, then the simulator $\mathcal{S}'$ for π' sends abort to the trusted party, outputs whatever $\mathcal{A}'$ outputs and halts. Otherwise, it invokes the simulator $\mathcal{S}$ that is guaranteed to exist for π on the residual adversary $\mathcal{A}$ that is obtained by running $\mathcal{A}'$ until the end of the first call to f_{condbc} (including $\mathcal{A}'$ receiving the corrupted parties' output bits from this call). Then, $\mathcal{S}'$ sends whatever $\mathcal{S}$ wishes to send to the trusted party, and outputs whatever $\mathcal{S}$ outputs. Since f_{condbc} behaves exactly like f_{bc} when $b = 1$ in the first phase, we have that the output distribution generated by $\mathcal{S}'$ is identical to that of $\mathcal{S}$ when $b = 1$. Furthermore, when $b = 0$, it is clear that the simulation is perfect. ■

4.2 Fairness with Identifiable Abort Implies Guaranteed Output Delivery

Before proceeding to prove that fairness implies guaranteed output delivery in a model with a broadcast channel, we first show that fairness with identifiable abort implies guaranteed output delivery. Recall that a protocol securely computes a functionality f with identifiable abort, if when the adversary causes an abort all honest parties receive $\perp$ as output along with the identity of a corrupted party. If a protocol securely computes f with fairness and identifiable abort, then it is guaranteed that if the adversary aborts, it learns nothing about the output and all honest parties learn an identity of a corrupted party. In this situation, the parties can eliminate the identified corrupted party and execute the protocol again, where an arbitrary party emulates the operations of the eliminated party using a default input. Since nothing was learned by the adversary when an abort occurs, the parties can rerun the protocol from scratch (without the identified corrupted party) and nothing more than a single output will be revealed to the adversary. Specifically, given a protocol π that computes f with fairness and identifiable abort, we can construct a new protocol π' that computes f with guaranteed output delivery. In the protocol π', the parties iteratively execute π, where in each iteration, either the adversary does not abort and all honest parties receive consistent output, or the adversary aborts without learning anything and the parties identify a corrupted party, who is eliminated from the next iteration.

Theorem 4.2. *Let f be an n-party functionality and let $t \leq n$. If f can be t-securely computed with fairness and identifiable abort, then f can be t-securely computed with guaranteed output delivery.*

Proof. We prove the theorem by constructing a protocol π that t-securely computes f with guaranteed output delivery in the $(f, \mathsf{id\text{-}fair})$-hybrid model. For every party P_i, we assign a default input value $\tilde{x}_i$ and construct the protocol π as follows:

1. Let $\mathcal{P}_1 = \{1, \ldots, n\}$ denote the set of indices of all participating parties.
2. For $i = 1, \ldots, t + 1$
   (a) All parties in $\mathcal{P}_i$ send their inputs to the trusted party computing f, where the party with the lowest index in $\mathcal{P}_i$ simulates all parties in $\mathcal{P}_1 \setminus \mathcal{P}_i$, using their predetermined default input values.
   For each $j \in \mathcal{P}_i$, denote the output of P_j from f by y_j.
   (b) For every $j \in \mathcal{P}_i$, party P_j checks if y_j is a valid output, if so P_j outputs y_j and halts. Otherwise all parties receive $(\perp, i^*)$ as output, where i^* is an index of a corrupted party. If $i^* \notin \mathcal{P}_i$ (and so i^* is a previously identified corrupted party), then all parties set i^* to be the party with the lowest index in $\mathcal{P}_i$.
   (c) Set $\mathcal{P}_{i+1} = \mathcal{P}_i \setminus \{i^*\}$.

First note that there are at most $t+1$ iterations; therefore π terminates in polynomial time. Let $\mathcal{A}$ be an adversary attacking π and let $\mathcal{I}$ be the set of corrupted parties. We construct a simulator $\mathcal{S}$ for the ideal model with f and guaranteed output delivery, as follows. $\mathcal{S}$ invokes $\mathcal{A}$ and receives its inputs to f in every iteration. If an iteration contains an abort, then $\mathcal{S}$ simulates sending the response $(\perp, i^*)$ to all parties, and proceeds to the next iteration. In the first iteration in which no abort is sent (and such an iteration must exist since there are $t+1$ iterations and in every iteration except for the last one corrupted party is removed), $\mathcal{S}$ sends the inputs of the corrupted parties that $\mathcal{A}$ sent to the trusted party computing f. In addition, $\mathcal{S}$ sends the values for any corrupted parties that were identified in previous iterations: if the lowest index remaining is honest, then $\mathcal{S}$ sets these values to be the default values; else, it sets these values to be the values sent by $\mathcal{A}$ for these parties. Upon receiving the output from its trusted party, $\mathcal{S}$ hands it to $\mathcal{A}$ as if it were the output of the corrupted parties in the iteration of π, and outputs whatever $\mathcal{A}$ outputs.

The simulation in the $(f, \text{id-fair})$-hybrid model is perfect since $\mathcal{S}$ can perfectly simulate the trusted party for all iterations in which an abort is sent. Furthermore, in the first iteration for which an abort is not sent, $\mathcal{S}$ sends f the exact inputs upon which the function f is computed in the protocol. Thus, the view of $\mathcal{A}$ and the output of the honest parties in the simulation with $\mathcal{S}$ are identical to their view and output in an execution of π in the $(f, \text{id-fair})$-hybrid model. ■

4.3 Fairness with Broadcast Implies Guaranteed Output Delivery

In Section 4.2, we saw that if a functionality can be securely computed with fairness and identifiable abort, then it can be securely computed with guaranteed output delivery. In this section, we show that assuming the existence of a broadcast channel, there is a protocol compiler that given a protocol computing a functionality f with fairness, outputs a protocol computing f with fairness and identifiable abort. Therefore, assuming broadcast, fairness implies guaranteed output delivery.

The protocol compiler we present is a modification of the GMW compiler, which relies on the code of the underlying fair protocol and requires non-black-box access to the protocol. (Therefore, this result does not contradict the proof in Section 5 that black-box access to an ideal functionality that computes f with fairness does not help to achieve guaranteed output delivery.) The underlying idea is to use the GMW compiler [GMW87, Gol04]. However, instead of enforcing semi-honest behaviour, the compiler is used in order to achieve security with identifiable abort. This is accomplished by tweaking the GMW compiler so that first only public-coin zero-knowledge proofs are used, and second if an honest party detects dishonest behaviour—i.e., if some party does not send a message or fails to provide a zero-knowledge proof for a message it sent—the honest parties record the identity i^* of the cheating party. We stress that the parties do *not* abort the protocol at this point, but rather continue until the end to see if they received $\perp$ or not. If they received $\perp$, then they output $(\perp, i^*)$ and halt. Else, if they received proper output, then they output it. Note that if the parties were to halt as soon as they detected a cheating party, then this would not be secure since it is possible that some of the corrupted parties already received output by that point. Thus, they conclude the protocol to determine whether they should abort or not.

The soundness of this method holds because in the GMW compiler with public-coin zero-knowledge proofs, a corrupted party cannot make an honest party fail, and all parties can verify if the zero-knowledge proof was successful or not. A brief description of the GMW compiler appears in Appendix A.7. We prove the following:

Theorem 4.3. *Assume the existence of one-way functions and let $t \leq n$. If a functionality f can be t-securely computed with fairness assuming a broadcast channel, then f can be t-securely computed with guaranteed output delivery.*

Proof: We begin by proving that fairness with a broadcast channel implies fairness with identifiable abort.

Lemma 4.4. *Assume the existence of one-way functions and let $t \leq n$. Then, there exists a polynomial-time protocol compiler that receives any protocol π, running over a broadcast channel, and outputs a protocol π', such that if π t-securely computes a functionality f with fairness then π' t-securely computes f with fairness and identifiable abort.*

Proof Sketch: Since the protocol is run over a single broadcasts channel, if at any point a party does not broadcast a message when it is suppose to, then all the parties detect it and can identify this party as corrupted.

We consider a tweaked version of the GMW compiler. The input-commitment phase and the coin-generation phase are kept the same, with the sole exception that if a party is identified a corrupted at this stage (e.g., if it does not send any value) then all the parties hard-wire to the function the default input value corresponding to this party. In the protocol-emulation phase, when a sender transmits a message to a receiver, they execute a strong zero-knowledge proof of knowledge with perfect completeness, in which the sender acts as the prover and the receiver as the verifier. The statement is that the message was constructed by the next-message function, based on the sender's input, random coins and the history of all the messages the sender received in the protocol. However, if the prover fails to prove the statement, unlike in the GMW compiler, the verifier does not immediately broadcast the verification coins, but stores the verification coins along with the identity of the sender in memory, and resumes the protocol.

At the end of the protocol emulation, each party checks if it received an output, if so it outputs it and halts. If a party did not receive an output and it received a message for which the corresponding zero-knowledge proof failed, it broadcasts the verification coins it used during the zero-knowledge proof. In this case, the other parties verify if this is a justified reject, and if so they output $\perp$ along with the identity of the prover. If the reject is not justified, the parties output $\perp$ along with the identity of the party that sent the false verification coins.

Since the zero-knowledge proof has perfect completeness, a corrupted party cannot produce verification coins that will falsely reject an honest party. Hence, only parties that deviate from the protocol can be identified as corrupted.

It case each honest party finishes the execution of the compiled protocol with some output, the compiled protocol remains secure, based on the security of the underlying protocol and of the zero-knowledge proof.

In case one of the honest parties did not get an output, there must be at least one message that does not meet the protocol's specification, hence at least one honest party received a message without a valid proof. Therefore, all the honest parties output $\perp$ along with an identity of a corrupted party. However, in this situation, the adversary does not learn anything about the output, since otherwise there exists an attack violating the fairness of the underlying protocol π. Hence, the compiled protocol retains fairness. ■

Applying Theorem 4.2 to Lemma 4.4 we have that f can be t-securely computed with guaranteed output delivery, completing the proof of the theorem. ■

5 Black-Box Fairness does not help for Guaranteed Output Delivery

In this section we show that the ability to securely compute a functionality with complete fairness does not assist in computing the functionality with guaranteed output delivery, at least in a black-box manner. More precisely, a functionality f can be securely computed with guaranteed output delivery in the (f, fair)-hybrid model if and only if f can be securely computed with guaranteed output delivery in the plain model.

The idea is simply that any protocol that provides guaranteed output delivery in the (f, fair)-hybrid model has to work even if the output of every call to the trusted party computing f fairly concludes with an abort. This is because a corrupted party can always send abort to the trusted party in every such call.

Proposition 5.1. *Let f be an n-party functionality and let $t \leq n$. Then, f can be t-securely computed in the (f, fair)-hybrid model with guaranteed output delivery if and only if f can be t-securely computed in the real model with guaranteed output delivery.*

Proof Sketch: If f can be t-securely computed in the real model with guaranteed output delivery, then clearly it can be t-securely computed in the (f, fair)-hybrid model with guaranteed output delivery by simply not sending anything to the trusted party.

For the other direction, let π be a protocol that t-securely computes f in the (f, fair)-hybrid model with guaranteed output delivery. We construct a protocol π' in the real model which operates exactly like π, except that whenever there is a call in π to the ideal functionality f, the parties in π' emulate receiving $\perp$ as output. It is immediate that for every adversary $\mathcal{A}'$ for π', there exists an adversary $\mathcal{A}$ for π so that the output distributions of the two executions are identical ($\mathcal{A}$ just sends abort to every ideal call in π, and otherwise sends the same messages that $\mathcal{A}'$ sends). By the assumption that π is secure, there exists a simulator $\mathcal{S}$ for the ideal model for f with guaranteed output delivery. This implies that $\mathcal{S}$ is also a good simulator for $\mathcal{A}'$ in π', and so π' t-securely computes f with guaranteed output delivery in the real model. ■

6 Additional Results

In this section we prove two additional results. First, there exist functionalities for which identifiable abort cannot be achieved (irrespective of fairness), and fairness and guaranteed output delivery are equivalent for fail-stop adversaries.

6.1 Identifiable Abort cannot be Achieved without Broadcast

We show that security with identifiable abort cannot be achieved in general without assuming a broadcast channel.

Proposition 6.1. *Assume the existence of one-way functions and let $t \geq n/3$. There exist functionalities that cannot be t-securely computed with identifiable abort, in the point-to-point network model.*

Proof Sketch: Assume by contradiction that the PKI setup functionality defined by

$$f_{\mathsf{PKI}}(\lambda, \ldots, \lambda) = ((\vec{pk}, sk_1), \ldots, (\vec{pk}, sk_n)),$$

can be t-securely computed with identifiable abort for $t = n/3$, where $\vec{pk} = (pk_1, \ldots, pk_n)$ and each (pk_i, sk_i) are a public/private key pair for a secure digital-signature scheme (that exists if one-way function exists). Then, we can t-securely compute f_{bc} by running the protocol π that is assumed to exist for f_{PKI}, where π is t-secure with identifiable abort. As in the proof of Theorem 4.2, if π ends with abort then the party who is identified as corrupted is removed. This continues iteratively until the protocol π terminates without abort, in which case a valid PKI is established between all remaining parties. Given this PKI, the parties can run authenticated broadcast in order to securely compute f_{bc}. Since f_{bc} cannot be securely computed for $t = n/3$, we have a contradiction. ■

6.2 Fairness Implies Guaranteed Output Delivery for Fail-Stop Adversaries

In the presence of malicious adversaries, fairness and guaranteed output delivery are different notions, since there exist functionalities that can be computed with complete fairness but cannot be computed with guaranteed output delivery. In the presence of semi-honest adversaries, it is immediate that both notions are equivalent, since the adversary cannot abort. In this section, we show that in the presence of the fail-stop adversaries, i.e., when the corrupted parties follow the protocol with the exception that the adversary is allowed to abort, fairness implies guaranteed output delivery.

The underlying idea is that if a corrupted party does not send a message to an honest party during the execution of a fair protocol, the honest party can inform all parties that it identified a corrupted party. Since the adversary is fail-stop, corrupted parties cannot lie and falsely incriminate an honest party. Similarly to the proof of Theorem 4.3, the parties do not halt if a party is detected cheating (i.e., halting early). Rather, the parties continue to the end of the protocol: if the protocol ended with output then they take the output and halt; otherwise, they remove the cheating party and begin again. Since the original protocol is fair, this guarantees that nothing is learned by any party if anyone receives abort; thus, they can safely run the protocol again. As in the proof of Theorem 4.2, this process is repeated iteratively until no abort is received. We conclude that:

Theorem 6.2. *Let f be an n-party functionality and let $t \leq n$. Then, f can be t-securely computed with fairness in the presence of fail-stop adversaries, if and only if f can be t-securely computed with guaranteed output delivery in the presence of fail-stop adversaries.*

References

[AL10] Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[Ash14] Gilad Asharov. Towards Characterizing Complete Fairness in Secure Two-Party Computation. In *Proceedings of the 11th Theory of Cryptography Conference, TCC 2014*, pages 291–316, 2014.

[Bea91] Donald Beaver. Foundations of Secure Interactive Computing. In *Advances in Cryptology – CRYPTO '91*, pages 377–391, 1991.

[Can00] Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[CL14] Ran Cohen and Yehuda Lindell. Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation. In *Advances in Cryptology – ASIACRYPT 2014, part II*, pages 466–485, 2014.

[Cle86] Richard Cleve. Limits on the Security of Coin Flips When Half the Processors are Faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369, 1986.

[FGH+02] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. Detectable Byzantine Agreement Secure against Faulty Majorities. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 118–126, 2002.

[FGMvR02] Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional Byzantine Agreement and Multi-party Computation Secure against Dishonest Minorities from Scratch. In *Advances in Cryptology – EUROCRYPT 2002*, pages 482–501, 2002.

[GHKL08] Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete Fairness in Secure Two-Party Computation. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–422, 2008.

[GK09] Dov Gordon and Jonathan Katz. Complete Fairness in Multi-party Computation without an Honest Majority. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 19–35, 2009.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Advances in Cryptology – CRYPTO '90*, pages 77–93, 1990.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure Multi-Party Computation without Agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[Kil91] Joe Kilian. A General Completeness Theorem for Two-Party Games. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 553–560, 1991.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[MR91] Silvio Micali and Phillip Rogaway. Secure Computation (Abstract). In *Advances in Cryptology – CRYPTO '91*, pages 392–404, 1991.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PW92] Birgit Pfitzmann and Michael Waidner. Unconditional Byzantine Agreement for any Number of Faulty Processors. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 339–350, 1992.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–85, 1989.

A Definitions and Preliminaries

Notations: We let $\kappa \in \mathbb{N}$ denote the security parameter. A function $\mathsf{negl}(\kappa)$ is negligible if for every positive polynomial $p(\kappa)$ and all sufficiently large $\kappa \in \mathbb{N}$ it holds that $\mathsf{negl}(\kappa) < 1/p(\kappa)$. A distribution ensemble $X = \{X_\kappa\}_{\kappa\in\mathbb{N}}$ is an infinite sequence of random variables indexed by $\kappa \in \mathbb{N}$. Two distribution ensembles $X = \{X(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a\in\{0,1\}^*,\kappa\in\mathbb{N}}$ are *computationally indistinguishable* (denoted $X \stackrel{\mathrm{c}}{\equiv} Y$) if for every non-uniform polynomial-time distinguisher $\mathcal{D}$ there exists a function $\mathsf{negl}(\kappa)$, such that for every $a \in \{0,1\}^*$ and all sufficiently large κ's

$$|\Pr[\mathcal{D}(X(a,\kappa),1^\kappa) = 1] - \Pr[\mathcal{D}(Y(a,\kappa),1^\kappa) = 1]| \leq \mathsf{negl}(\kappa).$$

Functionalities: An n-party functionality is a random process that maps vectors of n inputs to vectors of n outputs, denoted as $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where $f = (f_1, \ldots, f_n)$. That is, for a vector of inputs $\vec{x} = (x_1, \ldots, x_n)$, the output-vector is a random variable $(f_1(\vec{x}), \ldots, f_n(\vec{x}))$ ranging over vectors of strings. The output for the ith party (with input x_i) is defined to be $f_i(\vec{x})$. We denote an empty input by λ. In case of *symmetric functionalities*, where $f_1 = f_2 = \ldots = f_n$, by abuse of notation we refer to the functionality f as f_1.

All of the results in this paper apply to the case of *reactive functionalities*, which are multi-phase computations, e.g. commitment schemes. In this case, the functionality to be computed is modeled by a Turing machine that continually receives inputs and generates outputs. Our definition is based on function evaluation in order to simplify the presentation.

Adversarial behaviour: Loosely speaking, the aim of a secure multiparty protocol is to protect the honest parties against dishonest behavior from the corrupted parties. This is normally modeled using a central adversarial entity, which controls the set of corrupted parties and instructs them how to operate. That is, the adversary obtains the views of the corrupted parties, consisting of their inputs, random tapes and incoming messages, and provides them with the messages that they are to send in the execution of the protocol.

We differentiate between three types of adversaries:

- **Semi-honest adversaries:** a semi-honest adversary always instructs the corrupted parties to follow the protocol. Semi-honest adversaries model "honest but curious" behaviour, where

the adversary tries to learn additional information other than the output, based on the internal states of the corrupted parties.

- **Fail-stop adversaries:** a fail-stop adversary instructs the corrupted parties to follow the protocol as a semi-honest adversary, but it may also instruct a corrupted party to halt early (only sending some of its messages in a round).
- **Malicious adversaries:** a malicious adversary can instruct the corrupted parties to deviate from the protocol in any arbitrary way it chooses. There are no restrictions on the behaviour of malicious adversaries.

Unless stated otherwise, we consider malicious adversaries who may arbitrarily deviate from the protocol specification. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, parties may refuse to participate in the protocol, may substitute their local input (and enter with a different input) and may cease participating in the protocol before it terminates. Essentially, secure protocols limit the adversary to such behaviour only.

We further assume that the adversary is *computationally bounded* and *static*. By computationally bounded we mean that the adversary is modeled by a non-uniform probabilistic polynomial-time interactive Turing machine. By static, we mean that at the beginning of the execution, the adversary is given a set $\mathcal{I}$ of corrupted parties which it controls.

A.1 Security of Protocols

In this section we present definitions for secure multi-party computation. Out definitions follow [Gol04], which in turn follows [GL90, Bea91, MR91, Can00]. We consider several definitions of security: security with guaranteed output delivery, security with complete fairness, security with complete fairness and identifiable abort, security with abort and security with identifiable abort.

All of these security definitions are based on the *real/ideal paradigm*, i.e., comparing what an adversary can do in the real execution of the protocol to what it can do in an ideal model, where an uncorrupted trusted party assists the parties. In an ideal-model execution, each party sends its input to the trusted party over a perfectly secure channel, the trusted party computes the function based on these inputs and sends to each party its corresponding output. Informally, a protocol is secure if whatever an adversary can do in the real protocol (where no trusted party exists) can be done in the above-described ideal computation. The difference between the various security definitions is related to whether the ideal-model adversary is allowed to abort the ideal execution, and if so at what stage and under which conditions.

A.2 Execution in the Real World

We first define a real-model execution. In the real model, the parties execute the protocol in a synchronous network with rushing. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their message from this round) followed by a *receive phase* (where they receive messages from other parties). The adversary is *rushing* which means that it can see the messages the honest parties send in a round, before determining the messages that the corrupted parties send in that round.

We assume the parties are connected via a fully connected point-to-point network; we refer to this model as the point-to-point model. We sometimes assume that the parties are given access to a

physical broadcast channel in addition to the point-to-point network; we refer to this model as the broadcast model. In each section it will be explicitly clarified whether the existence of a broadcast channel is assumed or not. The communication lines between parties are assumed to be ideally authenticated but not private (and thus the adversary cannot modify messages sent between two honest parties but can read them).[3] Furthermore, the delivery of messages between honest parties is guaranteed. Finally, we note that we do not assume any trusted preprocessing phase (that can be used to setup a public-key infrastructure, for example).

Throughout a real execution, all the honest parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. Then, at the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted parties output nothing and the adversary outputs an (arbitrary) function of its view of the computation (which contains the views of all the corrupted parties). Without loss of generality, we assume that the adversary always outputs its view (and not some function of it).

Definition A.1 (real-model execution)**.** *Let f be an n-party functionality, let π be a multiparty protocol for computing f and let κ be the security parameter. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. Then, the* joint execution of π under $(\mathcal{A}, \mathcal{I})$ in the real model*, on input vector $\vec{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\vec{x}, \kappa)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the protocol interaction, where for every $i \in \mathcal{I}$, party P_i computes its messages according to $\mathcal{A}$, and for every $j \notin \mathcal{I}$, party P_j computes its messages according to π.*

A.3 Execution in the Ideal World

We consider several ideal worlds, each provides a different notion of security.

A.3.1 Secure Computation with Guaranteed Output Delivery

This definition provides the strongest notion of security we consider. According to this definition, the protocol can terminate only when all parties receive their prescribed output. Recall that a malicious party can always substitute its input or refuse to participate. Therefore, the ideal model takes this inherent adversarial behavior into account by giving the adversary the ability to do this also in the ideal model. Since the adversary cannot abort the execution of the protocol in this model, fail-stop adversaries are equivalent to semi-honest adversaries (in particular, they cannot substitute their input). An ideal execution proceeds as follows:

Send inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. Maliciously corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary. Denote by x'_i the value sent by P_i. In the case of a semi-honest or fail-stop adversary, we require that $x'_i = x_i$.

Trusted party answers the parties: If x'_i is outside of the domain for P_i or P_i sends no input, the trusted party sets x'_i to be some predetermined default value. Next, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every i.

[3] If private channels are needed, then privacy can be achieved over authenticated channels by simply using public-key encryption. This works for static corruptions and computationally bounded adversaries, as we consider in this work.

Outputs: Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary outputs an arbitrary function of the initial inputs $\{x_i\}_{i\in\mathcal{I}}$ and the messages received by the corrupted parties from the trusted party $\{y_i\}_{i\in\mathcal{I}}$.

Definition A.2 (ideal-model computation with guaranteed output delivery)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, where $f = (f_1,\ldots,f_n)$, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\vec{x} = (x_1,\ldots,x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\text{g.d.}}_{f,\mathcal{I},\mathcal{A}(z)}(\vec{x},\kappa)$, is defined as the output vector of $P_1,\ldots,P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

A.3.2 Secure Computation with Complete Fairness

This definition is similar to the previous one, except that the execution can terminate in two possible ways: the first is when all parties receive their prescribed output (as in the previous case) and the second is when all parties (including the corrupted parties) abort without receiving output. This is "fair" since in both cases the adversary receives no more information than the honest parties. In this definition, when sending the inputs to the trusted party, the adversary is allowed to send a special abort command. In this case, the trusted party sends a special abort symbol $\perp$ as the output to all parties. Without loss of generality, we assume that a malicious party always sends an input which is either in the corresponding input domain or abort, since in case the trusted party receives a value outside of the domain, it can proceed as if abort was sent. In this definition, fail-stop adversaries have the additional capability over semi-honest adversaries to abort the computation without anyone receiving output. An ideal execution proceeds as follows:

Send inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. Corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary. In addition, there exists a special abort input. Denote by x'_i the value sent by P_i. We require that in the case of a semi-honest adversary $x'_i = x_i$, whereas in the case of a fail-stop adversary $x'_i \in \{x_i, \text{abort}\}$.

Trusted party answers the parties: If there exists $i \in [n]$ such that $x'_i = \text{abort}$, the trusted party sends $\perp$ to all the parties. Otherwise, the trusted party computes $f(x'_1,\ldots,x'_n) = (y_1,\ldots,y_n)$ and sends y_i to party P_i for every $i \in [n]$.

Outputs: As in definition A.2.

Definition A.3 (ideal-model computation with complete fairness)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, where $f = (f_1,\ldots,f_n)$, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\vec{x} = (x_1,\ldots,x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\text{fair}}_{f,\mathcal{I},\mathcal{A}(z)}(\vec{x},\kappa)$, is defined as the output vector of $P_1,\ldots,P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

A.3.3 Secure Computation with Complete Fairness And Identifiable Abort

This definition, is identical to the previous definition of secure computation with complete fairness, except that if the adversary aborts the computation, all honest parties learn the identity of one of

the corrupted parties.

Send inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. Corrupted parties may send the trusted party arbitrary inputs as instructed by the adversary. Denote by x'_i the value sent by P_i. In case the adversary instructs P_i to send abort, it chooses an index of a corrupted party $i^* \in \mathcal{I}$ and sets $x'_i = (\mathsf{abort}, i^*)$. We require that in the case of a semi-honest adversary, $x'_i = x_i$, whereas in the case of a fail-stop adversary $x'_i \in \{x_i, (\mathsf{abort}, i^*)\}$.

Trusted party answers the parties: If there exists $i \in [n]$ such that $x'_i = (\mathsf{abort}, i^*)$ and $i^* \in \mathcal{I}$, the trusted party sends $(\perp, i^*)$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every $i \in [n]$.

Outputs: As in definition A.2.

Definition A.4 (ideal-model computation with complete fairness and identifiable abort)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, where $f = (f_1, \ldots, f_n)$, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\vec{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{fair,id\text{-}abort}}_{f,\mathcal{I},\mathcal{A}(z)}(\vec{x}, \kappa)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

A.3.4 Secure Computation with Abort

This definition is similar to secure computation with fairness, however the protocol can also terminate when corrupted parties receive output yet honest parties do not. However, if one honest party receives output then so do all honest parties. Thus, this is the notion of *unanimous abort*.

Send inputs to trusted party: As in Definition A.3.

Trusted party answers adversary: If there exists $i \in [n]$ such that $x'_i = \mathsf{abort}$, the trusted party sends $\perp$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every $i \in \mathcal{I}$.

Trusted party answers remaining parties: The adversary, depending on the views of all the corrupted parties, sends the trusted party either $\mathsf{continue}$ or abort. In case of $\mathsf{continue}$, the trusted party sends y_i to P_i for every $i \notin \mathcal{I}$, whereas in case of abort the trusted party sends $\perp$ to P_i for every $i \notin \mathcal{I}$.

Outputs: As in definitions A.2.

Definition A.5 (ideal-model computation with abort)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, where $f = (f_1, \ldots, f_n)$, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\vec{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{I},\mathcal{A}(z)}(\vec{x}, \kappa)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

A.3.5 Secure Computation with Identifiable Abort

This definition is identical to the previous definition of secure computation with abort, except that if the adversary aborts the computation, all honest parties learn the identity of one of the corrupted parties.

Send inputs to trusted party: As in Definition A.4.

Trusted party answers adversary: If there exists $i \in [n]$ such that $x'_i = (\mathsf{abort}, i^*)$ and $i^* \in \mathcal{I}$, the trusted party sends $(\perp, i^*)$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to every *corrupted* party P_i (for every $i \in \mathcal{I}$).

Trusted party answers remaining parties: The adversary, depending on the views of all the corrupted parties, sends the trusted party either $\mathsf{continue}$ or (abort, i^*), where $i^* \in \mathcal{I}$. In case of (abort, i^*) with $i^* \in \mathcal{I}$ the trusted party sends $(\perp, i^*)$ to P_i for every $i \notin \mathcal{I}$; in case of $\mathsf{continue}$, the trusted party sends y_i to P_i for every $i \notin \mathcal{I}$.

Outputs: As in definition A.2.

Definition A.6 (ideal-model computation with identifiable abort)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, where $f = (f_1, \ldots, f_n)$, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\vec{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{id\text{-}abort}}_{f,\mathcal{I},\mathcal{A}(z)}(\vec{x}, \kappa)$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

A.4 Security as Emulation of a Real Execution in the Ideal Model

Having defined the ideal and real models, we can now define security of protocols. The underlying idea of the definition is that the adversary can do no more harm in a real protocol execution than in the ideal model (where security trivially holds). This is formulated by saying that adversaries in the ideal model are able to simulate adversaries in an execution of a protocol in the real model.

Definition A.7. *Let $\mathsf{type} \in \{\mathsf{g.d.}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a protocol computing f. We say that* protocol π t-securely computes f with "type" *if for every non-uniform polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic (expected) polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| \leq t$,*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\vec{x}, \kappa)\right\}_{(\vec{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \stackrel{\text{c}}{\equiv} \left\{\text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathcal{S}(z)}(\vec{x}, \kappa)\right\}_{(\vec{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

A.5 The Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. The parties communicate with this trusted party in exactly the same way as in the ideal models described above. The question of which ideal model is considered must be specified. Specifically, the trusted party may work according to any of the ideal models that we have defined above.

Let f be a functionality. Then, an execution of a protocol π computing a functionality g in the f-hybrid model, involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f. It is essential that the invocations of f are done sequentially, meaning that before an invocation of f begins, the preceding invocation of f must finish. In particular, there is at most a single call to f per round, and no other messages are sent during any round in which f is called. This is especially important for reactive functionalities, where the calls to f are carried out in phases, and a new invocation of f cannot take place before all phases of the previous invocation complete. In addition, no other messages in π can be sent before f is completed. For example, if f computes the commitment functionality, then after the first call to f, computing the commit phase, another invocation of f cannot take place until the decommit phase of the first invocation is completed. (In this specific example, it typically won't be useful unless other messages can be sent between the commit and decommit phase. This can be overcome by not modeling the commitment as an ideal functionality. Alternatively, if the functionality allows for multiple commitments, then ordinary messages can be sent between the commit and decommit phase of a specific message by repeatedly committing and decommitting. This is an annoying technicality, but is nevertheless an inherent limitation of the sequential composition theorem of [Can00].)

Let $\mathsf{type} \in \{\mathsf{g.d.}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine with auxiliary input z and let $\mathcal{I} \subseteq [n]$ be the set of corrupted parties. We denote by $\text{HYBRID}^{f,\mathsf{type}}_{\pi,\mathcal{I},\mathcal{A}(z)}(\vec{x},\kappa)$ the random variable consisting of the view of the adversary and the output of the honest parties, following an execution of π with ideal calls to a trusted party computing f according to the ideal model "type", on input vector $\vec{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$, and security parameter κ. We call this the (f, type)-hybrid model.

The sequential composition theorem of [Can00] states the following. Let ρ be a protocol that securely computes f in the ideal model "type". Then, if a protocol π computes g in the (f, type)-hybrid model, then the protocol π^ρ, that is obtained from π by replacing all ideal calls to the trusted party computing f with the protocol ρ, securely computes g in the real model.

Proposition A.8 ([Can00]). *Let* $\mathsf{type}_1, \mathsf{type}_2 \in \{\mathsf{g.d.}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. *Let f be an n-party functionality. Let ρ be a protocol that t-securely computes f with* type_1*, and let π be a protocol that t-securely computes g with* type_2 *in the (f, type_1)-hybrid model. Then protocol π^ρ t-securely computes g with* type_2 *in the real model.*

A.6 Definitions of Specific Functionalities

We next define three functionalities that will be used throughout the paper. The first functionality we consider is the n-party broadcast functionality, $f_{\mathsf{bc}} : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where the sender P_1 has an input $x \in \{0,1\}^*$ while all other parties have the empty input λ (in plain English, this means that only the first party P_1 has input). The output of each party is x.

$$f_{\mathsf{bc}}(x, \lambda, \ldots, \lambda) = (x, \ldots, x).$$

The second functionality n-party Boolean OR, $f_{\mathsf{OR}} : \{0,1\}^n \to \{0,1\}^n$, where each party P_i has an input bit $x_i \in \{0,1\}$. The output of each party is the OR of all the inputs $x = x_1 \vee \ldots \vee x_n$.

$$f_{\mathsf{or}}(x_1, \ldots, x_n) = (x, \ldots, x) \quad \text{where} \quad x = x_1 \vee \ldots \vee x_n.$$

The third functionality is the majority functionality for three parties, $f_{\mathsf{maj}} : \{0,1\}^3 \to \{0,1\}^3$, where each party P_i has an input bit $x_i \in \{0,1\}$. The output of each party is the majority value of the input bits.

$$f_{\mathsf{maj}}(x_1, x_2, x_3) = (x, x, x) \quad \text{where} \quad x = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \wedge (x_2 \wedge x_3).$$

A.7 The GMW Compiler

Given a multiparty protocol, the GMW compiler consists of a pre-compiler and an authenticated-computation compiler:

- The pre-compiler produces a protocol that behaves as the original protocol, but instead of using a point-to-point network, all the communication is sent over a single broadcast channel. Each party generates a pair of keys for a public-key encryption scheme and broadcasts the encryption key. Next, every message is encrypted under the public key of the receiver and sent over the broadcast channel.

- The authenticated-computation compiler produces a protocol which may abort, but otherwise is enforced to behave as the input protocol. This compiler consists of an input-commitment phase, a coin-generation phase and a protocol-emulation phase.

  1. In the input-commitment phase, every party commits to its input towards all other parties.
  2. In the coin-generation phase, the parties jointly generate random tapes for each party. Each party receives its random tape and commitments for the random tapes of all other parties.
  3. In the protocol-emulation phase, the parties emulate the input protocol, where for each message, the sending party and the receiving party execute a zero-knowledge proof, proving that the message is produced by the next-message function based on the input, random tape and all prior messages.
     Note that a malicious party may abort the execution during this phase by not sending a message or providing an invalid proof. However, when using a public-coin proof, *all* parties can publicly verify if the proof is successful or not, and a corrupted party cannot cause an honest prover to fail.

The first part of the GMW compiler transforms any protocol running over a point-to-point communication network into a protocol running over a single broadcast channel, under the assumption that collections of trapdoor permutations exist (in order to obtain public-key encryption). Since we begin with a protocol that works over a broadcast channel, we can ignore this step.