Secure Computation Without a Broadcast Channel

Shafi Goldwasser* Yehuda Lindell†

March 29, 2002

Abstract

It has recently been shown that executions of authenticated Byzantine Agreement protocols in which more than a third of the parties are faulty, cannot be composed concurrently, in parallel, or even sequentially (where the latter is true for deterministic protocols). This result puts into question any usage of authenticated Byzantine agreement in a setting where many executions take place. In particular, this is true for the whole body of work of secure multi-party protocols in the case that 1/3 or more of the parties are faulty. Such protocols strongly rely on the extensive use of a broadcast channel, which is in turn realized using authenticated Byzantine Agreement. Essentially, this use of Byzantine Agreement cannot be eliminated, since the standard definitions of secure multiparty computation actually imply Byzantine agreement. Moreover, it is accepted folklore that the use of a broadcast channel is essential for achieving any *meaningful* secure multiparty computation, when 1/3 or more of the parties are faulty.

In this paper we show that this folklore is false. We mildly relax the definition of secure computation allowing abort, and show how this definition can be reached. In fact, we show that any protocol that is secure when run using a broadcast channel, can be transformed into a protocol that is secure under our relaxed definition and without a broadcast channel (or trusted preprocessing phase). We stress that our transformation holds for secure computation in both the information-theoretic and computational models. As a result we can securely compose multiple concurrent executions of multiparty protocols. The transformation replaces the use of Byzantine agreement by a weaker form of agreement which can be realized deterministically with $O(1)$ round complexity, and furthermore composes concurrently. The agreement provided is mild indeed: each honest player is guaranteed to either abort or receive the value which was broadcasted by an honest broadcaster. If the broadcaster is faulty all honest player which do not abort, agree on the same value. Nevertheless, as we show, this is sufficient for achieving significant secure computation.

1 Introduction

The general problem of multi-party protocols is as follows: A set of m users with private inputs wish to perform a joint function of their inputs, so that each user receives its output, and none of the users learn anything beyond that output. This encompasses computations as simple as coin flipping and agreement, and as complex as electronic voting, electronic auctions, electronic cash schemes, anonymous transactions, and private information retrieval schemes.

*Department of Computer Science and Applied Math, The Weizmann Institute of Science, Rehovot 76100, Israel. Email: shafi@wisdom.weizmann.ac.il

†Department of Computer Science and Applied Math, The Weizmann Institute of Science, Rehovot 76100, Israel. Email: lindell@wisdom.weizmann.ac.il

1.1 Ground rules of the 80's

This problem was initiated and heavily studied in the mid to late 80's, during which time the following ground rules were set.

Fault Model: The model of faults (in its stronger form) was accepted to be a coalition of t users who can be adaptively corrupted by an adversary during the computation. Furthermore, once a user is corrupted, it can deviate in an arbitrary manner from the prescribed protocol. A weaker adversarial model considers a static adversary, in which the set of corrupted parties is fixed ahead of time.

Broadcast: The ability to "broadcast" messages (if needed) was assumed as a primitive, where broadcast takes on the meaning of the Byzantine Generals problem [20]. Namely, an honest user can deliver the same message to all honest users at a given round, and a message "broadcast" by a faulty user will result in all honest users agreeing on a single value for that message. From results obtained largely by the distributed computing community, it was known that

1. For $t < n/3$, Byzantine agreement is possible by a deterministic protocol with round complexity $O(t)$ [23], and by a probabilistic protocol with expected round complexity $O(1)$ [10];
2. For $t \geq n/3$, agreement is achievable using a protocol for *authenticated* Byzantine agreement, in which a public-key infrastructure for digital signatures is used [23, 20]. (This public-key infrastructure is assumed to be setup in a trusted preprocessing phase.) We note that an information theoretic analogue also exists [24]. The round complexity of the above protocols is $O(t)$.

   Assuming broadcast as a primitive in a point-to-point network was seen as non-problematic. This is firstly due to the fact that the standard definitions of secure multiparty computation for $t < n/3$ imply Byzantine Agreement (and thus all the lower bounds of Byzantine Agreement automatically apply). Secondly, Byzantine Agreement is anyway achievable for all values of t (with the added requirement of a trusted preprocessing phase in the case of $t \geq n/3$).

Fairness: In addition to the obvious goals of correctness, privacy, and independence of inputs, fairness was also considered as a desirable goal. Fairness takes on different meanings for different values of t. We will single out a few forms of fairness as follows: "optimal fairness" guarantees that all honest users receive their correct outputs, and the adversary cannot disrupt the computation, "fairness" guarantees that if a faulty user gets its output then all honest users get their output, and "designated fairness" means that there is a designated party who is guaranteed to receive output if any faulty user gets its output. (Actually, if this designated party is honest, then "fairness" in the previous sense is guaranteed. On the other hand, if the party is faulty, then faulty parties may receive output while honest users do not.) In the last two cases, the protocol may abort without any outputs delivered.

1.2 What is possible and what is impossible

The 80's also yielded a seemingly clear picture of what is possible and what is impossible, as follows:

1. For $t < n/3$, secure multi-party protocols with optimal fairness, can be achieved from scratch in a point-to-point network [3, 9].

2. For $t < n/2$, secure multi-party protocols with optimal fairness can be achieved in a point to point network assuming a broadcast primitive or alternatively assuming pre-processing in which authenticated channels are set up for the purpose of achieving broadcast [25].

3. For $t \geq n/2$, secure multi-party protocols with designated fairness can be achieved in a point to point network assuming a broadcast protocol (as in case (2)), and in addition the existence of an oblivious transfer protocol [16]. We note that a protocol realizing oblivious transfer can be obtained if trapdoor functions exist. Some works attempting to provide fairness in some partial form (e.g., ensuring that the faulty player will progress at the same rate toward learning their output as the honest players) also appeared [26, 14, 17, 2].

For all of the above ranges of t, either one must assume perfectly secret pair-wise communication channels or the existence of one-way functions (with a set-up phase to exchange secret keys). We note that all of the above results consider a stand-alone execution of a multi-party protocol only.

1.3 Composing Executions of Protocols

Driven by the internet development, in the last few years the issue of multiple executions of cryptographic protocols that take place independently of each other has taken center stage. On the one hand, definitions of security for such settings have been developed. Most notably the "universal composition" framework of Canetti [5] guarantees that if a protocol is universally composable, then it can be safely run concurrently with executions of itself and other arbitrary protocols.

On the other hand, it was shown in several cases that protocols which were secure when considered in a stand alone setting, no longer remain secure when concurrent or parallel executions are considered. Byzantine Agreement (BA) is one such example. Lindell et al. [21] consider the case that the number of faulty parties exceeds a third of the total number of parties. Recall that this is the range that authenticated BA was assumed to provide an answer. They prove that any such protocol, with or without a pre-processing stage cannot be composed concurrently, or in parallel. Moreover, when the protocol is deterministic, even sequential composition is impossible.

This result puts in question the whole body of work regarding running secure multi-party protocols when more than a third of the parties are faulty, as such protocols rely heavily on the extensive use of broadcast. As broadcast is in turn realized by running authenticated BA, the impossibility of composing authenticated BA securely, stands in the way.

The use of Byzantine agreement cannot be eliminated, since the standard definitions of multiparty secure protocols actually imply Byzantine agreement. Thus, all lower bounds regarding Byzantine agreement apply to generic secure multiparty protocols. Moreover, it is accepted folklore that the use of broadcast is essential to achieve any *meaningful* secure multiparty computation, when more than 1/3 of the parties are faulty. In this paper we show that this is false. We define a relaxation of multi-party secure protocols which can be achieved, without the use of broadcast. In a nutshell, the relaxation allows some of the honest players to abort.

1.4 Our Results

We present a mild relaxation of what is meant by secure multiparty computation which can be achieved when the number of faults exceeds 1/3. Essentially the relaxation does not require all honest players to exit the protocol with an output as specified by the original computation. Instead, it allows an honest player to either receive its correct output or abort the protocol, but never to compute an incorrect output. We achieve this definition for all $t < n$, and without using a broadcast channel or trusted pre-processing stage. We stress that our result holds in both the information

theoretic and the computational models. For a slightly further relaxation of the definition, we also show a round-efficient transformation of any secure protocol that uses a broadcast channel into one that is secure under this more relaxed definition.

Note that in the range of $t \geq n/2$, it is impossible to achieve fairness in the sense that the honest players always receive output. Rather, abort by the honest parties (even if the corrupt parties do receive output) must be allowed.

A corollary of our result is the ability to tranform any protocol that uses a broadcast channel and composes concurrently or in parallel, into a protocol that can be run in the point-to-point network. We note that by the result of [21], protocols that use authenticated Byzantine Agreement in order to realize the broadcast channel *do not* compose in parallel. We also present a protocol for universally composable broadcast [5], and show how this is combined with known results [8] in order to obtain universally composable multiparty protocols for any $t < n$.

Protocols with Abort. For clarity and illustration of our result, as well as the ability to compare it to other work, we define several varieties of protocols with abort. The formal definitions will appear in Section 2.

1. *Secure computation with fair abort*: according to this definition, one of two cases can occur. Either all parties receive output or no parties receive output. Thus, the adversary can conduct a denial of service attack, but nothing else.

2. *Secure computation with abort:* this is the standard definition used when a half or less of the parties are honest. As in the previous definition, the adversary may disrupt the computation and cause the honest parties to output a special abort symbol $\perp$ (rather than their prescribed output). However, unlike above, the adversary may receive the faulty parties' outputs, even if the honest parties abort (and thus the abort is not "fair"). In particular, the protocol *specifies* a single party such that the following holds: if this party is honest, then either all parties abort or all parties receive correct output (i.e., computation with fair abort is achieved). On the other hand, if the specified party is corrupt, then the adversary receives the corrupted parties' outputs and can decide whether or not the honest parties all receive their correct output or all receive abort. We stress that in all cases, if one honest party aborts, then so do all honest parties (and thus all are aware of the fact that the protocol was under attack).

3. *Secure computation with designated abort:* this is a new definition that mildly relaxes the previous one. The only difference is that some honest parties may abort and some may receive the output (rather than having all honest parties abort whenever one does). There are actually two definitions here. In the first one, there is a specified party as above. The only difference is that if this party is corrupt, then it may *designate* which honest parties receive their output and which abort (in the case that the specified party is honest, computation with fair abort is achieved as in the previous definition). In the second definition, the adversary always receives the corrupted parties' outputs, and always has the ability to designate which honest parties output $\perp$ and which receive their prescribed output. We call this definition "secure computation with *strong designated abort*", since the adversary's ability to cause aborts is stronger.

Using this terminology, our result shows how to achieve secure computation with designated abort. Note that in secure computation with abort and with designated abort, fairness is nevertheless guaranteed in the case that a specific party (say, party P_1) is honest. On the other hand, if P_1 is corrupt, then the adversary may receive its output without the honest party receiving theirs. This

definition is desirable over a definition in which fairness is never guaranteed because there may be scenarios where one of the parties is "more trusted" than others. For example, the government (or election committee) in an election protocol may not be trusted when it comes to biasing the result or learning private inputs. However, this cannot be successfully carried out in a secure protocol (even one allowing designated abort). On the other hand, the government has little to gain by having only some of the participants receive output (except to completely disrupt the election). Therefore, in such a scenario, security with designated abort may be reasonable.

We note that in a similar fashion one can define *broadcast with abort* in which all honest users either abort or agree on the same value broadcasted (this is almost identical to weak Byzantine Generals), and *broadcast with designated abort* in which all honest parties which do not abort receive the same broadcasted value (and the adversary can designate which parties will receive abort and which will receive output).

1.5 Related Work

We have recently learned of two independent and concurrent results [12, 13] studying a problem similar to ours, although apparently for different motivation.[1] (It seems that their focus is on removing the preprocessing phase, while our concern is mainly with protocol composition. Nevertheless, both our and their results achieve both goals.)

In [12], Fitzi et al. study the question of multiparty computation in the case that the number of faults is $t < n/2$. They show that in this case, it is possible to achieve weak Byzantine agreement (where loosely speaking, either all honest parties abort or all honest parties agree on the broadcasted value). (We note that their protocol is probabilistic and "breaks" the $t < n/3$ lower-bound on deterministic weak Byzantine Agreement protocols of Lamport [19].) They further show that replacing the use of broadcast by Weak Byzantine agreement in secure computation protocols for $t < n/2$ such as [25], yields secure computation *with fair abort.* Namely, all honest parties either abort or all compute their correct output, and they agree on which output case occurred. Thus for the range of $t < n/2$ their solution achieves fairness whereas ours does not.

In subsequent work [13], Fitzi et al. studied the question of Byzantine agreement for any $t < n$ and whether its relaxation to weak Byzantine Agreement can be achieved without preprocessing. They show that it is indeed possible to achieve (randomized) weak Byzantine Agreement for *any* $t < n$, in $O(t)$ rounds. As above, this weak Byzantine Agreement protocol can be used instead of a broadcast channel in secure computation protocols such as [16, 2, 17], yielding secure computation *with abort* for any $t < n$.

In comparison, we achieve secure computation *with designated abort* for any $t < n$. The difference between our result and that of [13], is that in their work, either all parties abort or they all receive their correct output (even if the specified party is corrupted). On the other hand, in our case, if the specified party is faulty, then some honest parties may abort, while others receive the correct output (without necessarily knowing that others have aborted).

We conclude by comparing the round complexity of the secure protocols of [12, 13] with our results. They achieve secure computation by taking any secure protocol that uses a broadcast channel and replacing this channel with their protocol for weak Byzantine Agreement. Since the protocol for weak Byzantine Agreement requires $O(t)$ rounds, the result is an $O(t)$ multiplicative blow-up in the round complexity of the underlying secure protocol. On the other hand, we replace the broadcast channel with a simple 2-round protocol for broadcast with designated abort (to be exact, we add an additional blank round after each such protocol and therefore the round

[1] We were informed of this work while presenting our work at a seminar at MIT, February 14 2002.

complexity is 3 times that of the original protocol). Our protocols are therefore significantly more round efficient.[2] Finally we note that we can use the weak Byzantine Agreement protocol of [13] in order to take a generic r-round protocol for secure computation, and construct an $O(t+r)$-round protocol for secure computation with abort (rather than with designated abort). We therefore reduce the $O(tr)$ round complexity of [13] to $O(t+r)$, while achieving the same level of security.

2 Definitions

2.1 Secure Computation

In this section we present definitions for secure multiparty computation. The basic description and definitions are based on [15], which in turn follows [17, 1, 22, 4]. We actually consider a number of definitions here. In particular, we present formal definitions for secure computation with *fair abort,* with *abort,* with *designated abort,* and with *strong designated abort* (see Section 1.4). In addition, we refer to *secure computation without abort.* This is the standard definition used when more than half the parties are honest. According to this definition, all parties receive the output and the adversary cannot disrupt the computation. However, we will not formally present this definition here.

Notation: We denote by U_n the uniform distribution over $\{0,1\}^n$; for a set S we denote $s \in_R S$ when s is chosen uniformly from S; finally, computational indistinguishability is denoted by $\stackrel{\text{c}}{\equiv}$ and statistical closeness by $\stackrel{\text{s}}{\equiv}$.

Multiparty computation. A multiparty protocol problem (for m parties $P_1, \ldots, P_m$) is cast by specifying a random process that maps vectors of inputs to vectors of outputs (one for each party). We refer to such a process as an m-ary functionality and denote it $f : (\{0,1\}^*)^m \rightarrow (\{0,1\}^*)^m$, where $f = (f_1, \ldots, f_m)$. That is, for vector of inputs $\overline{x} = (x_1, \ldots, x_m)$, the output-vector is a random variable $(f_1(\overline{x}), \ldots, f_m(\overline{x}))$ ranging over vectors of strings. The i^{th} party (with input x_i) wishes to obtain $f_i(\overline{x})$. We often denote such a functionality by $(x_1, \ldots, x_m) \mapsto (f_1(\overline{x}), \ldots, f_m(\overline{x}))$.

Adversarial behavior. Loosely speaking, the aim of a secure multiparty protocol is to protect the honest parties against dishonest behavior from the corrupted parties. This "dishonest behavior" can manifest itself in a number of ways; in this paper we focus on *malicious* adversaries. Such an adversary may arbitrarily deviate from the specified protocol. 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.

Formally, the adversary is modeled by a non-uniform Turing machine: in the computational model this machine is polynomial-time whereas in the information-theoretic model it is unbounded. (We note that by standard arguments, we can assume that the adversary is deterministic.) For

[2] We note one subtle, yet important caveat. Given a *generic* protocol for secure computation that uses a broadcast channel and runs for r rounds, we obtain a $3r$-round protocol that is secure with designated abort (this is in contrast to the $O(tr)$ round complexity of [12, 13]). However, given a protocol that solves a specific secure computation problem, our transformation only achieves security with *strong* designated abort (see Section 1.4). In order to achieve security with designated abort, we must revert to a generic protocol. On the other hand, the transformation of [12, 13] works for any protocol. Thus, given a very efficient protocol for a specific problem, it may be "cheaper" to use [12, 13] rather than our results.

simplicity, in this work we consider a *static corruption* model. Therefore, at the beginning of the execution, the adversary is given a set I of corrupted parties which it controls. Then, throughout the computation, the adversary obtains the views of the corrupted parties, and provides them with the messages that they are to send.

Security of protocols (informal). The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an *ideal* computation involving an incorruptible *trusted party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted party exists) can do no more harm than if it was involved in the above-described ideal computation. We begin by formally defining this ideal computation.

2.1.1 Execution in the ideal model

The ideal model differs for each of the three definitions. We therefore present each one separately.

1. Secure computation with fair abort: Recall that a malicious party can always substitute its input or refuse to participate. Therefore, the ideal model must reflect these capabilities. We note that this definition is only achievable when the number of corrupted parties is less than $m/2$. An ideal execution proceeds as follows:

Inputs: Each party obtains its respective input from the input vector $\overline{x} = (x_1, \ldots, x_m)$.

Send inputs to trusted party: An honest party always sends its input x to the trusted party. The corrupted parties may, depending on their inputs $\{x_i\}_{i \in I}$, either abort or send modified $x'_i \in \{0,1\}^{|x_i|}$ to the trusted party. Denote the sequence of inputs obtained by the trusted party by $\overline{x}' = (x'_1, \ldots, x'_m)$ (for honest parties, $x' = x$ always).

Trusted party answers the parties: In case $\overline{x}'$ is a valid input sequence, the trusted party computes $f(\overline{x}')$ and sends $f_i(\overline{x}')$ to party P_i for every i. Otherwise (i.e., in case a corrupted party aborted or sent a non-valid input), the trusted party replies to all parties with a special symbol, $\perp$.

Outputs: An honest party always outputs the message that it received from the trusted party, whereas the corrupted parties output nothing (say, λ). On the other hand, the adversary outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in I}$ and the messages the corrupted parties obtained from the trusted party.

Definition 1 (ideal-model computation with fair abort): *Let $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be an m-ary functionality, where $f = (f_1, \ldots, f_m)$, and let $I \subset [m]$ be such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_m)$, denoted $\text{IDEAL}^{(1)}_{f,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_m$ and $\mathcal{A}$ resulting from the above described ideal process.*

2. Secure computation with abort: As before, a malicious party can always substitute its input or refuse to participate. However, when there are a half or less honest parties, it is not possible to continue computing in the case that the adversary ceases prematurely (this definition is usually used when the number of corrupted parties is not limited in any way). Thus, we cannot prevent the "early abort" phenomenon in which the adversary receives its output, whereas the honest parties do not receive theirs. This inherent limitation is therefore incorporated into the ideal execution. An ideal execution proceeds as follows:

Inputs: Each party obtains its respective input from the input vector $\overline{x} = (x_1, \ldots, x_m)$.

Send inputs to trusted party: An honest party always sends its input x to the trusted party. The corrupted parties may, depending on their inputs $\{x_i\}_{i \in I}$, either abort or send modified $x'_i \in \{0,1\}^{|x_i|}$ to the trusted party. Denote the sequence of inputs obtained by the trusted party by $\overline{x}' = (x'_1, \ldots, x'_m)$ (for honest parties, $x' = x$ always).

Trusted party answers first party: In case $\overline{x}'$ is a valid input sequence, the trusted party computes $f(\overline{x}')$ and sends $f_1(\overline{x}')$ to party P_1. Otherwise (i.e., in case a corrupted party aborted or sent a non-valid input), the trusted party replies to all parties with a special symbol, $\perp$.

Trusted party answers remaining parties: If the first party is not corrupted (i.e., $1 \notin I$), then the trusted party sends $f_j(\overline{x}')$ to party P_j, for *every* j.

In case the first party is corrupted, then for every $i \in I$, the trusted party sends $f_i(\overline{x})$ to party P_i (i.e., the corrupted parties receive their output first). Then, P_1, depending on the view of all the corrupted parties, instructs the trusted party to either send $f_j(\overline{x}')$ to P_j for every $j \notin I$, or to send $\perp$ to P_j for every $j \notin I$.

Outputs: An honest party always outputs the message that it received from the trusted party, whereas the corrupted parties output nothing (say, λ). On the other hand, the adversary outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in I}$ and the messages the corrupted parties obtained from the trusted party.

Definition 2 (ideal-model computation with abort): *Let $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be an m-ary functionality, where $f = (f_1, \ldots, f_m)$, and let $I \subset [m]$ be such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_m)$, denoted $\text{IDEAL}^{(2)}_{f,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_m$ and $\mathcal{A}$ resulting from the above described ideal process.*

3. Secure computation with designated abort: The definition here is almost the same as for secure computation with abort. The only difference is regarding the "trusted party answers remaining parties" item. In the above definition, all honest parties either receive their output or they receive $\perp$. Here, some of these parties may receive their (correct) output and some may receive $\perp$. In particular, if party P_1 is corrupted, then it may designate who does and does not receive output. We repeat only the relevant item:

Trusted party answers remaining parties: If the first party is not corrupted (i.e., $1 \notin I$), then the trusted party sends $f_j(\overline{x}')$ to party P_j, for *every* j.

In case the first party is corrupted, then for every $i \in I$, the trusted party sends $f_i(\overline{x}')$ to P_i (i.e., the corrupted parties receive their output first). Then, P_1, depending on the view of all the corrupted parties, chooses a subset of the honest parties $J \subseteq [m] \setminus I$ and sends J to the

trusted party. The trusted party then sends $f_j(\overline{x}')$ to P_j for every $j \in J$, and $\perp$ to all other parties.

Definition 3 (ideal-model computation with designated abort): *Let $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be an m-ary functionality, where $f = (f_1, \ldots, f_m)$, and let $I \subset [m]$ be such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_m)$, denoted $\text{IDEAL}^{(3)}_{f,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_m$ and $\mathcal{A}$ resulting from the above described ideal process.*

According to the above definition, the adversary can pinpoint any subset of parties J who should receive their output (with all others receiving $\perp$). We present the definition in this way for the sake of simplicity. However, it is possible to limit the adversary in the following way. The trusted party will send the outputs to the parties one by one (say, in order of P_1 to P_m). After each output has been sent, P_1 will tell the trusted party whether or not to send the next one. Once P_1 instructs the trusted party not to send an output, the trusted party sends all the remaining parties $\perp$ and halts. Thus, some honest parties may receive the output and some may receive $\perp$, however the adversary has very limited power in deciding who will receive what. Furthermore, consider a case where each party receives a different output (i.e., where the f_i's are distinct), and where the adversary controls P_i. Then, if the adversary wishes to receive $f_i(\overline{x})$, it must allow all parties $P_{i'}$ for $i' < i$ to also receive their correct output.

4. Secure computation with strong designated abort: This definition is very similar to the previous one, except that the first party P_1 does not receive the output first. Rather, the adversary first receives the output of the corrupted parties. Then, it designates which honest parties receive their output and which receive $\perp$. Formally,

Inputs: Each party obtains its respective input from the input vector $\overline{x} = (x_1, \ldots, x_m)$.

Send inputs to trusted party: An honest party always sends its input x to the trusted party. The corrupted parties may, depending on their inputs $\{x_i\}_{i \in I}$, either abort or send modified $x'_i \in \{0,1\}^{|x_i|}$ to the trusted party. Denote the sequence of inputs obtained by the trusted party by $\overline{x}' = (x'_1, \ldots, x'_m)$ (for honest parties, $x' = x$ always).

Trusted party answers adversary: In case $\overline{x}'$ is a valid input sequence, the trusted party computes $f(\overline{x}')$ and sends $f_i(\overline{x}')$ to party P_i for every $i \in I$. Otherwise (i.e., in case a corrupted party aborted or sent a non-valid input), the trusted party replies to all parties with a special symbol, $\perp$.

Trusted party answers remaining parties: The adversary, depending on the view of all the corrupted parties, chooses a subset of the honest parties $J \subseteq [m] \setminus I$ and sends J to the trusted party. The trusted party then sends $f_j(\overline{x}')$ to P_j for every $j \in J$, and $\perp$ to all other parties.

Outputs: An honest party always outputs the message that it received from the trusted party, whereas the corrupted parties output nothing (say, λ). On the other hand, the adversary outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in I}$ and the messages the corrupted parties obtained from the trusted party.

Definition 4 (ideal-model computation with strong designated abort): *Let $f : (\{0,1\}^*)^m \to (\{0,1\}^*)^m$ be an m-ary functionality, where $f = (f_1, \ldots, f_m)$, and let $I \subset [m]$ be such that for*

every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_m)$, denoted $\text{IDEAL}^{(4)}_{f,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_m$ and $\mathcal{A}$ resulting from the above described ideal process.*

2.1.2 Execution in the real model

We now 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). We stress that the messages sent by an honest party in a given round depend on the messages that it received in previous rounds only. On the other hand, the adversary can compute its messages in a given round based on the messages that it receives from the honest parties in the same round. The term rushing refers to this additional adversarial capability.

In this work, we consider a scenario where the parties are connected via a fully connected point-to-point network (and there is no broadcast channel). We refer to this model as the *point-to-point model* (in contrast to the *broadcast model* where the parties communicate via a physical broadcast channel). The communication lines between parties are assumed to be ideally authenticated and private (and thus the adversary cannot modify or read messages sent between two honest parties).[3] In the basic model, we assume that any message sent by an honest party to another honest party is received immediately. However, we also consider a model in which the adversary has control over the delivery of messages. That is, in every round, the adversary can decide to block (i.e., not deliver) some or all of the messages sent between the honest parties.[4] (We stress that since the communication lines are authenticated and private, the only thing that the adversary can do is prevent a message from being sent.) This model is of communication is the main model used by Canetti [5] in his work on universally composable security. Finally, we note that no preprocessing setup phase (such as a public-key infrastructure) is assumed.[5]

Throughout the execution, the honest parties all follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. Likewise, at the conclusion of the execution, the honest parties output their prescribed output from the protocol, whereas the corrupted parties output nothing. On the other hand, 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). Formally,

[3] We note that when the parties are assumed to be computationally bounded, privacy can be achieved over authenticated channels by using public-key encryption. Therefore, in such a setting, the requirement that the channels be private is not essential. However, we include it for simplicity.

[4] This capability models the following network scenario. All parties communicate on an open network, while encrypting and authenticating all messages sent. Therefore, the adversary cannot read or modify any message sent between honest parties. However, assume that the adversary has control over routing servers in the network. Then, it can always block communication (even if it cannot read or modify it).

[5] One can argue that achieving authenticated and private channels in practice essentially requires a preprocessing setup phase. Therefore, there is no reason not to utilize this preprocessing phase in the secure multiparty computation as well. In such a case, the preprocessing phase could be used in order to implement authenticated Byzantine Agreement (and thereby achieve secure broadcast for any number of faulty parties). However, we claim that the issue of achieving "secure communication channels" should be separated from the issue of secure multiparty computation. An example of why this is important was demonstrated in [21], who showed that authenticated Byzantine Agreement does not compose (in parallel or concurrently) when 2/3 or less of the parties are honest. On the other hand, secure channels can be achieved without any limitation on the protocol using them [7].

Definition 5 (real-model execution): *Let f be an m-ary functionality and let Π be a multiparty protocol for computing f. Furthermore, let $I \subset [m]$ be such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of Π under $(\mathcal{A}, I)$ in the real model *on input vector $\overline{x} = (x_1, \ldots, x_m)$, denoted $\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_m$ and $\mathcal{A}$ resulting from the protocol interaction, where for every $i \in I$, party P_i computes its messages according to $\mathcal{A}$, and for every $j \notin I$, party P_j computes its messages according to Π.*

2.1.3 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. Loosely speaking, the definition asserts that a secure multiparty protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate adversaries in an execution of a secure real-model protocol. The definition of security comes in two flavors. In the first, we consider polynomial-time bounded adversaries, and require that the simulation be such that the real-model and ideal-model output distributions are computationally indistinguishable. On the other hand, in the second, we consider unbounded adversaries and require that the simulation be such that the output distributions of the two models are statistically close.

Definition 6 (computational security): *Let f and Π be as above. We say that protocol Π is a protocol for* computational t-secure computation with fair abort *(resp.,* with abort*,* with designated abort *or* with strong designated abort*), if for every non-uniform polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [m]$ with $|I| < t$,*

$$\{\text{IDEAL}^{(\alpha)}_{f,(\mathcal{S},I)}(\overline{x})\}_{n\in\mathsf{N},\overline{x}\in(\{0,1\}^n)^m} \stackrel{\text{c}}{\equiv} \{\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x})\}_{n\in\mathsf{N},\overline{x}\in(\{0,1\}^n)^m}$$

where the value of $\alpha \in \{1,2,3,4\}$ depends on whether secure computation with fair abort, with abort, with designated abort or with strong designated abort is being considered.

Definition 7 (information-theoretic security): *Let f and Π be as above. We say that protocol Π is a protocol for* information-theoretic t-secure computation with fair abort *(resp.,* with abort*,* with designated abort *or* with strong designated abort*), if for every non-uniform adversary $\mathcal{A}$ for the real model, there exists a non-uniform adversary $\mathcal{S} = \{S_n\}$ for the ideal model such that for every $I \subset [m]$ with $|I| < t$,*

$$\{\text{IDEAL}^{(\alpha)}_{f,(\mathcal{S},I)}(\overline{x})\}_{n\in\mathsf{N},\overline{x}\in(\{0,1\}^n)^m} \stackrel{\text{s}}{\equiv} \{\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x})\}_{n\in\mathsf{N},\overline{x}\in(\{0,1\}^n)^m}$$

where the value of $\alpha \in \{1,2,3,4\}$ depends on whether secure computation with fair abort, with abort, with designated abort or with strong designated abort is being considered.

3 Broadcast with Designated Abort

Definition. In this section, we present a weak variant of the Byzantine Generals problem, that we call *"broadcast with designated abort"*. The main idea is to weaken both the agreement and validity requirements so that parties may output either the required value x or a special abort symbol $\perp$. However, there is no requirement regarding who should output x and who should output $\perp$. In particular, this means that the adversary can *designate* the parties who will output $\perp$. Formally,

Definition 8 (broadcast with designated abort): *Let $P_1, \ldots, P_m$, be m parties and let P_1 be the designated party* (*or* dealer) *with input x. In addition there is an adversary who controls up to t of the parties* (*which may include P_1*). *A protocol solves the* broadcast with designated abort *problem if the following three properties hold:*

1. Agreement: *If an honest party outputs x', then all honest parties output either x' or $\perp$.*
2. Validity: *If P_1 is honest, then all honest parties output either x or $\perp$.*
3. Non-triviality: *If all parties are honest, then all parties output x.*

The non-triviality requirement is needed to rule out a protocol in which all parties simply output $\perp$ and halt.

A protocol. We now present a simple protocol that solves the broadcast with designated abort problem for *any* t. As we will see later, despite its simplicity, this is enough for obtaining secure computation with designated abort.

Protocol 1 (broadcast with designated abort):

- **Input:** *P_1 has a value x to broadcast.*
- **The Protocol:**
  1. *P_1 sends x to all parties.*
  2. *Denote by x^i the value received by P_i in the previous round. Then, every party P_i (for $i > 1$) sends its value x^i to all other parties.*
  3. *Denote the value received by P_i from P_j in the previous round by x^i_j (recall that x^i denotes the value P_i received from P_1 in the first round). Then, P_i outputs x^i if this is the only value that it saw (i.e., if $x^i = x^i_2 = \cdots = x^i_n$). Otherwise, it outputs $\perp$.*
     We note that if P_i did not receive any value in the first round, then it always outputs $\perp$.

We now prove that Protocol 1 is secure, for any number of corrupted parties. That is,

Proposition 3.1 *Protocol* 1 *solves the broadcast with designated abort problem, for* any *number of corrupted parties t.*

Proof: The fact that the non-triviality condition is fulfilled is immediate. We now prove the other two conditions:

1. *Agreement:* Let P_i be an honest party, such that P_i outputs a value x'. Then, it must be that P_i received x' from P_1 in the first round (i.e., $x^i = x'$). Therefore, P_i sent this value to all other parties in the second round. Now, a party P_j will output x^j if this is the only value that it saw during the execution. However, as we have just seen, P_j definitely saw x' in the second round. Thus, P_j will only output x^j if $x^j = x'$. On the other hand, if P_j does not output x^j, then it outputs $\perp$.
2. *Validity:* If P_1 is honest, then all parties receive x in the first round. Therefore, they will only output x or $\perp$.

This completes the proof. ■

Notice that although the adversary is rather limited in that it can only cause parties to output a correct value or $\perp$, it does have the ability to choose exactly which honest parties will output the correct value and which will output $\perp$. Thus, it can indeed "designate" the set of honest parties which output $\perp$.

3.1 Strengthening Broadcast with Designated Abort

A natural question to ask is whether or not we can strengthen Definition 8 in one of the following two ways (and still obtain a protocol for $t \geq m/3$):

1. *Strengthen the agreement requirement:* If an honest party outputs a value x', then all honest parties output x'. (On the other hand, the validity requirement remains unchanged.)

2. *Strengthen the validity requirement:* If P_1 is honest, then all honest parties output x. (On the other hand, the agreement requirement remains unchanged.)

It is easy to see that the above strengthening of the agreement requirement results in the definition of weak Byzantine Generals. (The validity and non-triviality requirements combined together are equivalent to the validity requirement of weak Byzantine Generals.) Therefore, there exists no *deterministic* protocol for the case of $t \geq m/3$.

On the other hand, the strengthening of the validity requirement does *not* appear to imply weak Byzantine Generals or Agreement. (The main reason being that agreement must always hold for the weak Byzantine problems. Here, on the other hand, the agreement requirement does not necessarily hold for corrupted dealers.) Nevertheless, we show that no protocol exists for this problem for $t \geq m/3$. For shorthand, we call this stronger version of the problem "strong broadcast with designated abort".

Proposition 3.2 *There does not exist a protocol for the* strong *broadcast with designated abort problem, for $t \geq m/3$.*

The proof of Proposition 3.2 can be found in Appendix B.

We note that this problem is quite natural and can be viewed as the flip-side of the weak Byzantine Generals problem. (In that problem the validity condition is weakened and the agreement condition remains as in the original problem. On the other hand, here the agreement condition is slightly weakened and the validity condition is as in the original problem.) Such a protocol would allow an honest dealer to always successfully broadcast its value. On the other hand, a dishonest dealer can cause the honest parties to not fully agree. (This capability is however limited; the honest parties never output different non-$\perp$ values.) We conclude by noting that if the agreement condition is completely weakened so that it only holds when all parties are honest, then the resulting definition is easily obtained by simply having the dealer send its input to all parties, who then output the value that they receive.

3.2 Universally Composable Broadcast

In this section, we consider a different real model in which the adversary has control over the delivery of messages. We note that because the network is synchronous, the adversary cannot deliver a message late (i.e., in round r when it was sent in round r' and $r > r'$). The only capability that it is given is to *block* messages sent between honest parties.

In this section we show that it is possible to realize an ideal broadcast functionality in a universally composable way, within the above-described model. Essentially this means that any protocol that uses a broadcast channel, can be implemented without a broadcast channel and the result is the same. An important corollary of this result is the existence of universally composable multiparty computation with $t \geq m/3$, in a point-to-point network. This corollary is obtained by applying our secure realization of broadcast with known universally composable multiparty protocols. See Appendix A for an overview of the universal composition framework.

We begin by defining the ideal broadcast functionality which we wish to realize. This is defined in Figure 1.

Functionality $\mathcal{F}_{\text{BC}}$

$\mathcal{F}_{\text{BC}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$:

Upon receiving a message $(\mathsf{broadcast}, sid, x)$ from P_i, send $(\mathsf{broadcast}, sid, P_i, x)$ to all parties and to $\mathcal{S}$.

Figure 1: The ideal broadcast functionality

We now restate Protocol 1 in a way that is suitable for the universally composable framework.

Protocol 2 (universally composable broadcast):

- **Input:** *P_i has a value x to broadcast.*
- **The Protocol:**
  1. *P_i sends x to all parties.*
  2. *Denote by x^j the value received by P_j in the previous round. Then, every party P_j (for $j \neq i$) sends its value x^j to all other parties.*
  3. *Denote the value received by P_j from P_k in the previous round by x_k^j (recall that x^j denotes the value P_j received from P_i in the first round). Then, P_j outputs $(\mathsf{broadcast}, sid, P_i, x^j)$ if this is the only value that it saw (i.e., if $x^j = x_2^j = \cdots = x_n^j$). Otherwise, it outputs nothing.*

     We note that if P_j did not receive any value in the first round, then it always outputs nothing.

The result of this section is the following proposition:

Proposition 3.3 *Protocol 2 securely realizes $\mathcal{F}_{\text{BC}}$ (in the universally composable framework).*

Proof: Let $\mathcal{A}$ be a real-model adversary attacking Protocol 1. We construct an ideal-model adversary $\mathcal{S}$ for $\mathcal{F}_{\text{BC}}$ as follows. We differentiate between two cases: in the first the dealer is corrupted (and thus is controlled by $\mathcal{A}$), and in the second it is honest. Let P_i be the party broadcasting in this execution.

- *Case 1 – P_i is corrupt:* In the first round, $\mathcal{A}$ (controlling P_i) sends messages to ℓ of the honest parties for some ℓ; denote these by $x_1, \ldots, x_\ell$. Simulator $\mathcal{S}$ receives all these messages and then simulates the messages sent by the honest parties in the second round. Furthermore, $\mathcal{S}$ obtains all the messages sent by $\mathcal{A}$ in the second round.

Now, if there exist j and k such that $x_j \neq x_k$, then $\mathcal{S}$ sends nothing to $\mathcal{F}_{\text{BC}}$. Otherwise, let x be the message sent by $\mathcal{A}$. Then, $\mathcal{S}$ sends x to the ideal functionality $\mathcal{F}_{\text{BC}}$. Next, $\mathcal{S}$ defines the set of honest parties J who $\mathcal{A}$ had sent x to in the first round, and who did not receive any message $x' \neq x$ in the second round from $\mathcal{A}$. Then, $\mathcal{S}$ delivers the messages from the $\mathcal{F}_{\text{BC}}$ functionality to the parties in J, and only these parties.

- *Case* 2 – P_i *is honest:* $\mathcal{S}$ receives $(\mathsf{broadcast}, sid, P_i, x)$ from $\mathcal{F}_{\text{BC}}$ and simulates P_i's sending x to all the parties controlled by $\mathcal{A}$. Then, $\mathcal{S}$ receives back messages sent by $\mathcal{A}$ to the honest parties. $\mathcal{S}$ defines a set J of parties who $\mathcal{A}$ sends only x to in the second round. Then, $\mathcal{S}$ delivers the messages from the $\mathcal{F}_{\text{BC}}$ functionality to these and only these parties.

We claim that the global output of an ideal execution with $\mathcal{S}$ is *identically distributed* to the global output of a real execution with $\mathcal{A}$. We first deal with the case that P_i is corrupt. Now, if $\mathcal{A}$ sends two different messages in round 1 (i.e., if there exist j and k such that $x_j \neq x_k$), then by the protocol definition, all honest parties will see both x_j and x_k. Therefore, in a real execution all honest parties will output nothing. This is identical to the case that $\mathcal{S}$ does not send anything to $\mathcal{F}_{\text{BC}}$ in an ideal execution. On the other hand, if $\mathcal{A}$ sends the same message x to all honest parties in the first round, then the outputs depend on what $\mathcal{A}$ sends in the second round. (In this case, the honest parties all send x to each other in the second round and therefore these messages are of no consequence.) However, $\mathcal{S}$ receives all these messages from $\mathcal{A}$ and can therefore see which parties would output x and which parties would output nothing. Since $\mathcal{S}$ delivers the $(\mathsf{broadcast}, ...)$ messages from $\mathcal{F}_{\text{BC}}$ only to the parties which would output x in the real model, the output is identical.

In the case that P_i is honest, $\mathcal{A}$ can cause honest parties to output nothing (rather than x) by sending them messages $x' \neq x$ in the second round. As above, $\mathcal{S}$ receives all these messages and therefore its delivery of $(\mathsf{broadcast}, ...)$ messages from $\mathcal{F}_{\text{BC}}$ accurately represents exactly what happens in a real execution. ■

Our first corollary relates to the scenario where a majority of the parties are honest, but this majority may be less than 2/3 (i.e., $m/2 \leq t \leq 2m/3$). In this scenario, Canetti [5] showed that universally composable protocols exist for any functionality, assuming that the parties interact in a synchronous network with a broadcast channel. Combining this with Proposition 3.3 we have the following:

Corollary 9 *Consider a synchronous point-to-point network and assume that trapdoor permutations exist. Then, for any multiparty ideal functionality $\mathcal{F}$, there exists a protocol* Π *that securely realizes $\mathcal{F}$ in the in the presence of* malicious, static *adversaries, and for $t < m/2$ corruptions.*

The next corollary relates to a setting with an honest *minority*. In this setting, a common reference string is essential for achieving universal composability [6, 8]. Nevertheless, this is a weaker setup assumption than that of a public-key infrastructure (as required for authenticated Byzantine Agreement). Canetti et al. show in [8] that in a synchronous network with a *broadcast channel*, it is possible to securely compute any functionality. Therefore, by combining this with Proposition 3.3, we have that:

Corollary 10 *Consider a synchronous point-to-point network and assume that trapdoor permutations exist. Then, for any multiparty ideal functionality $\mathcal{F}$, there exists a protocol* Π *in the common reference string model, that securely realizes $\mathcal{F}$ in the presence of* malicious, static *adversaries, and for any number of corruptions.*

The above corollaries are actually less dramatic than they initially seem. This is because the universally composable framework (in its current definition) assumes the existence of unique session-

identifiers for every execution. Therefore, it is possible to use authenticated Byzantine Agreement in order to securely realize $\mathcal{F}_{\text{BC}}$ ([21] show that unique session identifiers suffice for composing authenticated Byzantine Agreement). Thus, the above corollaries refer to the possibility of achieving this without setup assumptions. They do not, however, enable us to achieve something that was previously unknown within the universally composability framework.[6] Nevertheless, Proposition 3.3 can be used to transform *any* protocol that uses a broadcast channel and composes in parallel or concurrently, into a protocol for the point-to-point model that still composes. This is *not* possible using authenticated Byzantine Agreement, as this does not compose [21]. We therefore conclude that in the model where the adversary controls message delivery, composition can be achieved that previously was not known. In particular, the protocol of Rabin and Ben-Or [25] for $m/2$-secure computation uses a broadcast channel and composes concurrently.[7] We therefore have,

Corollary 11 *Consider a synchronous point-to-point network where the adversary controls message delivery (and no unique session identifiers are assumed). Then, for any probabilistic polynomial-time m-ary functionality f, there exists a protocol for the information-theoretic $m/2$-secure computation of f, that composes concurrently.*

4 Secure Computation with Strong Designated Abort

In this section, we show that any protocol for secure computation (with abort or with fair abort) that is designed using a broadcast channel can be "compiled" into a protocol for secure computation with strong designated abort. Furthermore, the fault tolerance of the compiled protocol is the same as the original one. Actually, we require that the protocol to be compiled has the following property: the round in which a party sets its output is fixed in the protocol definition (and is not a random variable depending on the execution). We say that a protocol with this property has *fixed-round outputs*. The result of this section is formally stated in the following theorem:

Theorem 12 *There exists a (polynomial-time) protocol compiler that receives any protocol Π (with fixed-round outputs) for the* broadcast model *for input, and outputs a protocol Π' for the* point-to-point model *such that the following holds: If Π is a protocol for information-theoretic (resp., computational) t-secure computation (with abort or with fair abort), then Π' is a protocol for information-theoretic (resp., computational) t-secure computation with strong designated abort.*

Combining Theorem 12 with known protocols (specifically, [25] and [16][8]), we obtain the following corollaries:

Corollary 13 (information-theoretic security – compilation of [25]): *For any probabilistic polynomial-time m-ary functionality f, there exists a protocol in the point-to-point model, for the information-theoretic $m/2$-secure computation of f with strong designated abort.*

We note that this result is optimal in the following sense. Ben-Or, Goldwasser and Wigderson [3] showed that there are functions for which there do not exist information-theoretically private protocols when $t \geq m/2$. The definition of a "private" (rather than "secure") protocol, is regarding the

[6] In private communication with Ran Canetti, he agreed that unique session identifiers can be removed from the universal composability definition. In such a case, universally composable broadcast could not be achieved using authenticated Byzantine Agreement.

[7] We note that the fact that [25] composes concurrently has not been formally proved anywhere. Nevertheless, it is claimed in [5].

[8] We note that both the protocols of [25] and [16] have fixed-round outputs

behavior of corrupted parties. In a private protocol, corrupted parties follow the protocol specification, but attempt to learn more information than intended (such adversarial behavior is known as passive or semi-honest). Therefore, security with strong designated abort implies privacy (with fair abort), and this means that for information-theoretic security, resilience of $t \geq m/2$ is not possible.

Corollary 14 (computational security – compilation of [16]): *For any probabilistic polynomial-time m-ary functionality f, there exists a protocol in the point-to-point model, for the computational t-secure computation of f with strong designated abort, for* any *t.*

We now proceed to prove Theorem 12.

Proof of Theorem 12: Intuitively, we construct a protocol for the point-to-point model from a protocol for the broadcast model, by having the parties in the point-to-point network simulate the broadcast channel. When considering "pure" broadcast (i.e., Byzantine Generals), this is not possible for $t \geq m/3$. However, it is enough for us to simulate the broadcast channel using a protocol for "broadcast with designated abort". Recall that in such a protocol, either the correct value is delivered to all parties, or some parties output $\perp$. The idea is to halt the computation in the case that any honest party receives $\perp$ from a broadcast execution. The point at which the computation halts dictates which parties (if any) receive output. The key point is that if no honest party receives $\perp$, then the broadcast with designated abort protocol perfectly simulates a broadcast channel. Therefore, the result is that the original protocol (for the broadcast channel) is simulated perfectly until the point that it may prematurely halt.

Construction 3 (protocol compiler): *Given a protocol Π, the compiler produces a protocol Π'. The specification of the protocol Π' is as follows:*

- *The parties use Protocol* 1 *in order to emulate each broadcast message of protocol Π. Each round of Π is expanded into* 3 *rounds in Π': Protocol* 1 *is run in the first* 2 *rounds, and the third round is a* blank *round (the purpose of this third round will become clear later). The parties emulate Π according to the following instructions:*

  1. Broadcasting messages: *Let P_i be a party who is supposed to send a message m in the j^{th} round of Π. Then, in the j^{th} broadcast simulation of Π', all parties run an execution of Protocol* 1 *in which P_i plays the dealer role and sends m.*
  2. Receiving messages: *For each message that party P_i is supposed to receive from a broadcast in Π, party P_i participates in an execution of Protocol* 1 *as a receiver. If its output from this execution is a message m, then it appends m to its view (to be used for determining its later steps according to Π).*

     If it receives $\perp$ from this execution, then it sends $\perp$ to all parties in the next round (this round is the blank round following the execution of Protocol 1*), and halts immediately.*
  3. Blank rounds: *If a party P_i receives $\perp$ in a blank round, then it sends $\perp$ to all parties in the next blank round and halts, outputting $\perp$. Party P_i does* not *participate in the broadcast that precedes the next blank round. (We note that if this is the last round of the execution, then P_i simply halts.)*
  4. Output: *Let party P_i be such that it just received a non-$\perp$ value from a broadcast execution of Protocol* 1*. Furthermore, according to Π, at this point it is supposed to output its value*

(*denote this value* y). *Then,* P_i *waits for the next round* (*which is blank*)*: if it receives* $\perp$ *in this blank round, then it outputs* $\perp$*; otherwise, it outputs* y.[9]

In order to prove that Π' is t-secure with designated abort, we first define a different transformation of Π to $\tilde{\Pi}$ which is a hybrid protocol between Π and Π'. In particular, $\tilde{\Pi}$ is still run in the broadcast model. However, it provides corrupted parties with the additional ability of prematurely halting honest parties. We now define this hybrid protocol $\tilde{\Pi}$ and show that it is t-secure with designated abort:

Lemma 4.1 *Let* Π *be a protocol in the broadcast model that is computational* (*resp., information-theoretic*) t*-secure with abort or with fair abort. Then, define protocol* $\tilde{\Pi}$ (*also for the broadcast model*) *as follows:*

1. *Following each round of broadcast of* Π*, add a blank round.*
2. *If in a blank round, the message* $(P_i, \perp)$ *is broadcast, then* P_i *broadcasts* $(P_j, \perp)$ *for all* $j \neq i$ *in the next blank round and halts.* P_i *also does not broadcast any message in the next broadcast round* (*that precedes the next blank round*)*.*
3. *Apart from the above, the parties follow the instructions of* Π*.*
4. Output: *Let party* P_i *be such that it sets its output in round* r *of* Π*. Then, if it receives* $(P_i, \perp)$ *in the* s^{th} *blank round of* $\tilde{\Pi}$ *for any* $s \leq r$*, then it outputs* $\perp$*. Otherwise, it outputs its prescribed output.*

Then, $\tilde{\Pi}$ *is computational* (*resp., information-theoretic*) t*-secure with designated abort.*

Proof: We prove this theorem for the case that Π is computationally t-secure with abort. The other cases (information theoretic security and security with fair abort) are proved in the same way. Let $\tilde{\mathcal{A}}$ be a real-model adversary attacking $\tilde{\Pi}$. We begin by constructing a real-model adversary $\mathcal{A}$ attacking Π who causes the output distribution in Π to be very similar to that of $\tilde{\Pi}$ when under attack by $\tilde{\mathcal{A}}$. We do this because $\mathcal{A}$ is guaranteed to have a simulator $\mathcal{S}$ (by the security of Π), and we can therefore use $\mathcal{S}$ in order to construct a simulator $\tilde{\mathcal{S}}$ for $\tilde{\mathcal{A}}$. Adversary $\mathcal{A}$ receives the input sequence $\{x_i\}_{i\in I}$ and a series of random-tapes $\{r_i\}_{i\in I}$ (where r_i is the random-tape of P_i) and works as follows: $\mathcal{A}$ invokes $\tilde{\mathcal{A}}$ upon input $\{x_i\}_{i\in I}$ and random-tapes $\{r_i\}_{i\in I}$, and broadcasts all of its broadcast messages. Likewise, all messages received by $\mathcal{A}$ are internally forwarded to $\tilde{\mathcal{A}}$. Now, if $\tilde{\mathcal{A}}$ broadcasts $(P_i, \perp)$ for some honest P_i in any blank round (preceding the last blank round), then in the next broadcast round of Π, adversary $\mathcal{A}$ receives the broadcast messages from all the honest parties. $\mathcal{A}$ forwards to $\tilde{\mathcal{A}}$ only the messages from honest parties P_j for which $(P_j, \perp)$ was not sent by $\tilde{\mathcal{A}}$ in the previous blank round, and broadcasts whatever $\tilde{\mathcal{A}}$ does. Finally, $\mathcal{A}$ simulates for $\tilde{\mathcal{A}}$ the blank-round messages as sent by the honest parties in such a case, and halts. When $\mathcal{A}$ halts, it outputs whatever $\tilde{\mathcal{A}}$ does (which by assumption equals $\tilde{\mathcal{A}}$'s view of the execution). This completes the description of $\mathcal{A}$.

Before proceeding, we show that the only difference between an execution of Π with $\mathcal{A}$, and $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$, is that some additional honest parties may output $\perp$ in the execution of $\tilde{\Pi}$. That is, we

[9] The motivation behind this seemingly strange output procedure is as follows. If any honest party received $\perp$ from an execution of Protocol 1, then we would like all honest parties to disregard any messages received in this execution. Now, if an honest party receives $\perp$ from a broadcast, then it announces this to all parties in the next blank round. Thus, if an honest party receives $\perp$ in a blank round, it disregards the messages it received in the previous broadcast.

claim that the joint distribution of the outputs of all parties not outputting $\perp$ and the adversary, is identical in Π and $\tilde{\Pi}$. Formally, denote by $\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r})$, the output of an execution of Π with adversary $\mathcal{A}$, inputs $\overline{x}$, and random-tapes $\overline{r}$ (i.e., $\overline{r} = (r_1,\ldots,r_m)$ where P_i receives random-tape r_i). (Thus, $\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x}) = \{\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x},U_{|\overline{r}|})\}$.) Furthermore, for some set $J \subseteq [m] \setminus I$, denote by $\text{REAL}^{J}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r})$, the output of parties $\{P_j\}_{j\in J}$ in an execution of Π with $\mathcal{A}$, inputs $\overline{x}$ and random-tapes $\overline{r}$. Similarly, denote by $\text{REAL}^{\mathcal{A}}_{\Pi,(\mathcal{A},I)}(\overline{x})$, the output of adversary $\mathcal{A}$. Now, in any execution of $\tilde{\Pi}$, it is possible to divide the parties into those who output $\perp$ and those who do not output $\perp$. Let $J_{\text{REAL}} = J_{\text{REAL}}(\overline{x},U_{|\overline{r}|})$ be a random variable that takes values over subsets of parties in an execution of $\tilde{\Pi}$ with adversary $\tilde{\mathcal{A}}$, where for every $j \in J_{\text{REAL}}$, party P_j does *not* output $\perp$. (We stress that for every fixed $\overline{x}$ and $\overline{r}$, the set J_{REAL} is a fixed subset of $[m] \setminus I$. Furthermore, for every adversary $\tilde{\mathcal{A}}$ there is a different random variable J_{REAL}.)

Then, we claim that for any adversary $\tilde{\mathcal{A}}$ and set of corrupted parties I, and for any set of inputs and random-tapes $\overline{x}$ and $\overline{r}$,

$$\left(\text{REAL}^{\mathcal{A}}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r}), \text{REAL}^{J_{\text{REAL}}(\overline{x},\overline{r})}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r})\right) = \left(\text{REAL}^{\tilde{\mathcal{A}}}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x},\overline{r}), \text{REAL}^{J_{\text{REAL}}(\overline{x},\overline{r})}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x},\overline{r})\right) \quad (1)$$

where $\mathcal{A}$ is as defined above. Eq. (1) is shown as follows: Fix the input vector $\overline{x}$ and series of random tapes $\overline{r}$ (below, when we refer to an execution of Π or $\tilde{\Pi}$, we mean with respect to these fixed inputs and random-tapes). Now, let ρ be the blank-round number in which $\tilde{\mathcal{A}}$ first sends a $(P_i,\perp)$ message. Then, the messages sent and received by $\mathcal{A}$, $\tilde{\mathcal{A}}$ and all the honest parties in the first ρ broadcast rounds are identical in the executions of Π and $\tilde{\Pi}$ (recall that the blank rounds follow the broadcast rounds). Therefore, if an honest party sets its output by round ρ and does not output $\perp$, then its output is identical in Π and $\tilde{\Pi}$.[10] On the other hand, any honest party P_i in $\tilde{\Pi}$ who sets its output in round $\rho+1$ or later, outputs $\perp$ in $\tilde{\Pi}$ and therefore is not relevant to Eq. (1) (i.e., $i \notin J_{\text{REAL}}$). It remains to show that the outputs of $\tilde{\mathcal{A}}$ and $\mathcal{A}$ are also identical. However, this follows because $\mathcal{A}$ forwards to $\tilde{\mathcal{A}}$ exactly the messages that it expects to see. That is, for any party P_i for which $(P_i,\perp)$ was sent in round ρ, adversary $\tilde{\mathcal{A}}$ does not receive any message from P_i in broadcast round $\rho+1$. On the other hand, $\tilde{\mathcal{A}}$ receives the exact messages sent by the other parties (who only receive a $(P,\perp)$ message in blank-round $\rho+1$ from P_i). Finally, all parties halt after blank round $\rho+1$ in an execution of $\tilde{\Pi}$ and this is what $\tilde{\mathcal{A}}$ sees in $\mathcal{A}$'s execution of Π. This completes the proof of Eq. (1). We stress that the parties who output $\perp$ in $\tilde{\mathcal{A}}$ may have very different outputs in $\mathcal{A}$ (and in particular may output their prescribed outputs). Nevertheless, at this stage we are only interested in those parties not outputting $\perp$.

As we mentioned above, the reason that we defined $\mathcal{A}$ as above is that we can use the simulator guaranteed to exist for $\mathcal{A}$ in order to simulate $\tilde{\mathcal{A}}$. We now proceed to construct the simulator $\tilde{\mathcal{S}}$ for $\tilde{\mathcal{A}}$.

Simulator $\tilde{\mathcal{S}}$: The simulator $\tilde{\mathcal{S}}$ invokes $\mathcal{S}$ on $\mathcal{A}$ and sends the trusted party the corrupted parties' inputs exactly as sent by $\mathcal{S}$. Once $\tilde{\mathcal{S}}$ has forwarded these inputs to the trusted party, it receives all the corrupted parties' outputs (by the definition of security with strong designated abort). Therefore, $\mathcal{S}$ provides $\tilde{\mathcal{S}}$ with these outputs. Now, in the case that P_1 is not corrupted, $\mathcal{S}$ concludes at this point outputting some string. On the other hand, if P_1 is corrupted, then $\mathcal{S}$ first instructs the trusted party to send the honest parties $\perp$ or their prescribed outputs before concluding. $\tilde{\mathcal{S}}$ records this message and fixes its output to be whatever $\mathcal{S}$ outputs.

[10] We note that a party P_j who sets its output in round ρ may or may not output $\perp$, depending on the execution. In particular, if $(P_j,\perp)$ was sent in the r^{th} blank round by $\tilde{\mathcal{A}}$, then P_j outputs $\perp$. On the other hand, if it was only sent in the $r{+}1^{\text{th}}$ round, then P_j will output its prescribed output.

It remains to define the set J_{IDEAL} that $\tilde{\mathcal{S}}$ sends to the trusted party in the "trusted party answers remaining parties" stage of the ideal execution (recall that all honest parties in J_{IDEAL} receive their output and all others receive $\perp$). First notice that the string output by $\mathcal{S}$ is computationally indistinguishable to $\mathcal{A}$'s output from a real execution. However, by the definition of $\mathcal{A}$, this output contains $\tilde{\mathcal{A}}$'s view of an execution of $\tilde{\Pi}$. Furthermore, $\tilde{\mathcal{A}}$'s view fully defines which honest parties in an execution of $\tilde{\Pi}$ output $\perp$ and which receive their output. Therefore, $\tilde{\mathcal{S}}$ examines this view and defines the set J_{IDEAL} accordingly. We first introduce some notation. Let ρ be the first blank round in which a $(P_i, \perp)$ message is sent in the view output by $\mathcal{S}$. Furthermore, denote by P_ρ the set of parties P_i who received $(P_i, \perp)$ in round ρ. Notice that by the definition of $\tilde{\Pi}$, all honest parties are included in the set $P_\rho \cup P_{\rho+1}$. (In the case that ρ is the last round of the execution, then $P_{\rho+1}$ is defined to be empty. This is because in this case, only the parties in P_ρ ever receive $\perp$.)

We begin with the case that ρ is undefined: that is, when $\tilde{\mathcal{A}}$ does not send any $(P_i, \perp)$ messages during the execution. In such a case, $\tilde{\mathcal{S}}$ defines J_{IDEAL} in the following way: if $\mathcal{S}$ instructs the trusted party to send $\perp$ to all the honest parties, then $\tilde{\mathcal{S}}$ defines $J_{\text{IDEAL}} = \phi$. Otherwise, all honest parties receive their correct outputs, and therefore $\tilde{\mathcal{S}}$ defines $J_{\text{IDEAL}} = [m] \setminus I$. We now proceed to the case that a $(P_i, \perp)$ message is sent during the execution. In this case, $\tilde{\mathcal{S}}$ defines J_{IDEAL} to equal all the parties in P_ρ whose outputs are set prior to broadcast round ρ along with all the parties in $P_{\rho+1}$ whose outputs are set prior to broadcast round $\rho+1$.[11] Once J_{IDEAL} is defined, $\tilde{\mathcal{S}}$ sends it to the trusted party and halts. This completes the description of $\tilde{\mathcal{S}}$.

We now wish to show that the output of an ideal execution with designated abort with $\tilde{\mathcal{S}}$ is computationally indistinguishable to the output of a real execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$. We begin by showing an analog to Eq. (1). That is, we show that the outputs of parties not outputting $\perp$ are the same in an ideal execution of Π and an ideal execution of $\tilde{\Pi}$. Formally, we claim the following: For every set I, every set of inputs $\overline{x}$ and every random-tape r (for $\mathcal{S}$ or $\tilde{\mathcal{S}}$),

$$\left(\text{IDEAL}^{(4)\ \tilde{S}}_{f,(\tilde{\mathcal{S}},I)}(\overline{x},\overline{r}), \text{IDEAL}^{(4)\ J_{\text{IDEAL}}(\overline{x},r)}_{f,(\tilde{\mathcal{S}},I)}(\overline{x},r)\right) = \left(\text{IDEAL}^{(2)\ \mathcal{S}}_{f,(\mathcal{S},I)}(\overline{x},r), \text{IDEAL}^{(2)\ J_{\text{IDEAL}}(\overline{x},r)}_{f,(\mathcal{S},I)}(\overline{x},r)\right) \quad (2)$$

where $\mathcal{S}$ and $\tilde{\mathcal{S}}$ are invoked with inputs $\overline{x}$ and random-tape r, and where $J_{\text{IDEAL}}(\overline{x}, r)$ equals the set of parties in the ideal execution with $\tilde{\mathcal{S}}$ who do not output $\perp$. (I.e., $J_{\text{IDEAL}}(\overline{x}, r)$ equals the set J_{IDEAL} sent by $\tilde{\mathcal{S}}$ to the trusted party when the input vector equals $\overline{x}$ and its random-tape equals r.) In order to see that Eq. (2) holds, notice the following. Firstly, $\tilde{\mathcal{S}}$ (upon input $\{x_i\}_{i\in I}$ and random-tape r) sends exactly the same inputs to the trusted party as $\mathcal{S}$ does (upon input $\{x_i\}_{i\in I}$ and random-tape r). Now, the outputs of all honest parties not outputting $\perp$ are fixed by $\overline{x}$ and the inputs sent to the trusted party by the simulators $\mathcal{S}$ or $\tilde{\mathcal{S}}$. Therefore, if $\mathcal{S}$ and $\tilde{\mathcal{S}}$ send the same inputs, it follows that all parties not outputting $\perp$ have exactly the same output. Secondly, $\tilde{\mathcal{S}}$ outputs exactly the same string that $\mathcal{S}$ outputs. Eq. (2) follows.

Now, Π is computationally t-secure with abort. It therefore holds that for every set $I \subset [m]$ such that $|I| < t$, and for every set $J \subseteq [m]$

$$\left\{\text{IDEAL}^{(2)\ \mathcal{S}}_{f,(\mathcal{S},I)}(\overline{x}), \text{IDEAL}^{(2)\ J}_{f,(\mathcal{S},I)}(\overline{x})\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^{\mathcal{A}}_{\Pi,(\mathcal{A},I)}(\overline{x}), \text{REAL}^{J}_{\Pi,(\mathcal{A},I)}(\overline{x})\right\}$$

However, notice that the sets J_{REAL} and J_{IDEAL} are fully defined given $\tilde{\mathcal{A}}$ and $\tilde{\mathcal{S}}$'s outputs respectively. By the definitions of $\mathcal{A}$ and $\mathcal{S}$, it follows that their outputs also fully define J_{REAL} and J_{IDEAL}.

[11] It is for this point of the proof that we require the protocol Π to be such that the round in which a party sets its output is fixed in the protocol definition. Otherwise, the adversary's view may not be enough to determine when an honest party sets its output.

Therefore, it holds that,

$$\left\{\text{IDEAL}^{(2)\ \mathcal{S}}_{f,(\mathcal{S},I)}(\overline{x}), \text{IDEAL}^{(2)\ J_{\text{IDEAL}}(\overline{x})}_{f,(\mathcal{S},I)}(\overline{x})\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^{\mathcal{A}}_{\Pi,(\mathcal{A},I)}(\overline{x}), \text{REAL}^{J_{\text{REAL}}(\overline{x})}_{\Pi,(\mathcal{A},I)}(\overline{x})\right\} \tag{3}$$

Combining Eq. (3) with Equations (1) and (2), we have that

$$\left\{\text{IDEAL}^{(4)\ \tilde{\mathcal{S}}}_{f,(\tilde{\mathcal{S}},I)}(\overline{x}), \text{IDEAL}^{(4)\ J_{\text{IDEAL}}(\overline{x})}_{f,(\tilde{\mathcal{S}},I)}(\overline{x})\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}^{\tilde{\mathcal{A}}}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x}), \text{REAL}^{J_{\text{REAL}}(\overline{x})}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x})\right\} \tag{4}$$

It remains to show that the entire output distributions (including the honest parties *not* in J) are computationally indistinguishable. However, this is immediate, because for every party P_i for which $i \notin J$, it holds that P_i outputs $\perp$ (this is true for both the real and ideal executions). Therefore,

$$\left\{\text{IDEAL}^{(4)}_{f,(\tilde{\mathcal{S}},I)}(\overline{x})\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x})\right\}$$

completing the proof of Lemma 4.1. ■

Recall that our aim is to show the security of the compiled protocol Π' (and not $\tilde{\Pi}$). However, intuitively, there is no difference between $\tilde{\Pi}$ and Π'. The reason is as follows: in $\tilde{\Pi}$, the adversary can instruct any honest party P_i to ignore a broadcast by sending it $(P_i, \perp)$ in the following blank round. On the other hand, in Π', the same effect can be achieved by having the "broadcast with designated abort" terminate with P_i receiving $\perp$. Therefore, whatever an adversary attacking Π' can achieve, an adversary attacking $\tilde{\Pi}$ can also achieve. Formally:

Lemma 4.2 *Let Π be a protocol in the broadcast model that is information-theoretic or computational t-secure with abort or with fair abort, and let $\tilde{\Pi}$ be the transformation of Π as described in Lemma 4.1. Then, for every real-model adversary $\mathcal{A}'$ for Π' of Construction 3, there exists a real-model adversary $\tilde{\mathcal{A}}$ for $\tilde{\Pi}$, such that for every $I \subset [m]$ with $|I| < t$,*

$$\left\{\text{REAL}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x})\right\}_{n\in\mathsf{N},\overline{x}\in(\{0,1\}^n)^m} \equiv \left\{\text{REAL}_{\Pi',(\mathcal{A}',I)}(\overline{x})\right\}_{n\in\mathsf{N},\overline{x}\in(\{0,1\}^n)^m}$$

Proof: We begin by describing the adversary $\tilde{\mathcal{A}}$. Adversary $\tilde{\mathcal{A}}$ invokes $\mathcal{A}'$ and in every round of the execution of $\tilde{\Pi}$ works as follows:

- *Broadcast round r:*
  1. Receive the broadcast messages from the honest parties. For every message broadcast by an honest party, simulate a "broadcast with designated abort" execution, playing the honest parties' roles (where $\mathcal{A}'$ plays the corrupted parties' roles).
  2. Play the honest parties' roles in "broadcast with designated abort" executions, in which $\mathcal{A}'$ sends messages to the honest parties. Consider a particular execution in which a corrupted party P plays the dealer. If all the honest parties receive $\perp$ in this execution, then $\tilde{\mathcal{A}}$ broadcasts nothing. On the other hand, if at least one honest party outputs a message m, then $\tilde{\mathcal{A}}$ broadcasts m.
- *Blank round r:* Let B denote the set of honest parties receiving $\perp$ in any of the above simulated "broadcast with designated abort" executions (i.e., for broadcast round r). Then, for every $P_i \in B$, adversary $\tilde{\mathcal{A}}$ broadcasts $(P_i, \perp)$ in this blank round.

At the conclusion of the execution, $\tilde{A}$ outputs whatever $\mathcal{A}'$ does. This completes the description of $\tilde{\mathcal{A}}$. The fact that $\tilde{\mathcal{A}}$ perfectly simulates an execution of Π' with $\mathcal{A}'$ follows directly from the definition of Π'. That is, the only difference between $\tilde{\Pi}$ and Π' is that in Π' the broadcast channel is replaced by Protocol 1. This means that some parties may receive $\perp$ instead of the broadcasted message. However, in this case, $\tilde{\mathcal{A}}$ knows exactly who these parties are and can send them $\perp$ in the following blank round. Furthermore, the simulation of the executions of Protocol 1 itself by $\tilde{\mathcal{A}}$ is perfect. Therefore, the outputs of all the honest parties and $\tilde{\mathcal{A}}$ in $\tilde{\Pi}$, are identically distributed to the outputs of the honest parties and $\mathcal{A}'$ in an execution of Π'. ■

Now, let $\mathcal{A}'$ be an adversary attacking Π'. By Lemma 4.2, we have that there exists an adversary $\tilde{\mathcal{A}}$ attacking $\tilde{\Pi}$ such that the output distributions of Π' with $\mathcal{A}'$, and $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$ are identical. Then, by Lemma 4.1, we have that for real-model adversary $\tilde{\mathcal{A}}$ for $\tilde{\Pi}$, there exists an ideal-model simulator $\tilde{\mathcal{S}}$ such that the output distributions of a real execution with $\tilde{\mathcal{A}}$ and an ideal execution with designated abort with $\tilde{\mathcal{S}}$ are computationally indistinguishable (or statistically close). We conclude that the output distribution of a real execution of Π' with adversary $\mathcal{A}'$ is computationally indistinguishable (or statistically close) to an ideal execution with designated abort with $\tilde{\mathcal{S}}$. That is, Π' is t-secure with designated abort, as required. ■

The complexity of protocol $\tilde{\Pi}$: We remark that the transformation of Π to $\tilde{\Pi}$ preserves the round complexity of Π. In particular, the number of rounds in $\tilde{\Pi}$ equals exactly 3 times the number of rounds in Π. On the other hand, the bandwidth of $\tilde{\Pi}$ is the same as that of Π except for the cost incurred in simulating the broadcast channel. Notice that in the "broadcast with designated abort" protocol, if a dealer sends an n-bit message, then the total bandwidth equals $m \cdot n$. (If the dealer cheats and sends different messages, then the bandwidth is upper-bound by the length of the longest message times m.) Therefore, the number of bits sent in an execution of $\tilde{\Pi}$ is only m times that sent in an execution of Π. (However, a subtle point to notice here is that the adversary can cause honest parties to send very long messages. This is because in Π, only the adversary sends its messages on the broadcast channel. On the other hand, in $\tilde{\Pi}$, every message sent by the adversary is repeated by all honest parties.)

5 Secure Computation with Designated Abort

In this section, we present the main results of this paper. That is, we show that there for any functionality f, there exists a protocol for *computational* t-secure computation of f with designated abort, for any t. (This construction assumes the existence of trapdoor permutations.) Furthermore, for any functionality f, there exists a protocol for *information-theoretic* $m/2$-secure computation of f with designated abort (and without any assumptions).

Outline: We begin by motivating why the strategy used to obtain secure computation with strong designated abort is not enough here. The problem lies in the fact that due to the use of a "broadcast with designated abort" protocol (and not a real broadcast channel), the adversary can disrupt communication between honest parties. That is, of course, unless this communication need not be broadcast. Now, in the definition of security with designated abort, once an honest P_1 receives its output, it must be able to give this output to all honest parties. That is, the adversary must not be allowed to disrupt the communication, following the time that an honest P_1 receives its output. This means that using a broadcast channel in the final stage where the remaining parties receive their outputs is problematic.

We solve this problem here by having the parties compute a different functionality. This functionality is such that when P_1 gets its output, it can supply all the other parties with their output directly and without broadcast. On the other hand, P_1 itself should learn nothing of the other parties' outputs. As a first attempt, consider the following. Instead of computing $\overline{x} \mapsto (f_1(\overline{x}), \ldots, f_m(\overline{x}))$, the parties first compute the following:

$$\overline{x} = (x_1, \ldots, x_m) \mapsto ((f_1(\overline{x}), r_2, \ldots, r_m), f_2(\overline{x}) \oplus r_2, \ldots, f_m(\overline{x}) \oplus r_m)$$

Then, the parties use a protocol that is secure with *strong* designated abort in order to compute the new functionality. Once they have finished, all parties announce whether or not they have received their output. If any of them announces $\perp$, then P_1 sends all parties $\perp$ and halts. Otherwise, P_1 individually sends r_i to P_i for every i, allowing all parties to reconstruct their correct output.

However, this strategy is flawed. The problem is that a cheating P_1 can send the honest parties wrong values and thus cause them to conclude with incorrect outputs. This is in contradiction to what is required of all secure protocols. Therefore, the functionality we use is one in which a corrupt P_1 cannot cheat. In particular, we require a scheme with the following properties. The scheme has a setup phase in which an honest dealer provides two parties P_1 and P_i with strings r_1 and r_2. The strings are uniformly distributed when viewed independently. However, given them both, it is possible to extract some predetermined value (in this case, it will be P_i's output). Finally, P_1 has no choice but to provide P_i with r_1 itself; any modification to r_1 will result in detection by P_i with overwhelming probability. We note that the trusted dealer will be replaced by a protocol that is secure with *strong* designated abort. We now describe such a scheme:

Construction 4 (binded sharing of a bit between A and B):

- Trusted dealer's input: *A bit b.*
- Setup Phase:
  1. *The dealer chooses n triplets (r_i^1, r_i^2, r_i^3), where each $r_i^j \in_R \{0,1\}$ is uniform under the constraint that $r_i^1 \oplus r_i^2 \oplus r_i^3 = b$.*
  2. *The dealer gives A the pair (r_i^1, r_i^2) for every i.*
  3. *The dealer gives B the pair (r_i^j, r_i^3) and the index j, where $j \in_R \{1,2\}$. That is, in every triple, B receives the 3^{rd} bit and another bit which is randomly chosen as the first or second.*
- Reveal Phase:
  1. *A sends B its pairs (r_i^1, r_i^2) for every i.*
  2. *For every pair received, B checks that r_i^j as it received, equals the bit sent by A (B received the index j in the setup phase so can check this). If this holds, then B checks that for every i, $r_i^1 \oplus r_i^2 \oplus r_i^3$ equals the same bit $b' \in \{0,1\}$. If this holds, then B outputs b'. Otherwise it outputs $\perp$.*

We note that binded sharing of a string can be achieved by separately sharing each bit. Now, it is immediate that neither A or B learn anything about b from the setup phase (i.e., privacy after the setup phase is information theoretic). On the other hand, assume that A attempts to have B output $b' \neq b$. This requires A to flip one bit in every triple (otherwise, if there is one triple which A does not touch, then B will receive b from that triple and will never output $b' \neq b$). Now, the probability that B will not detect A's bit flip in a single triple equals $1/2$. Therefore, the probability that B will not detect A's bit flip in every triple equals 2^{-n}, which is negligible.

Finally, we note that if A were to know which bits B received in the setup phase, then it could easily have B output any bit in the reveal phase. Thus, the scheme has a trapdoor-type property. We need this for our protocol. We note that the above construction is reminiscent of those found in [18].

Theorem 15 *For any probabilistic polynomial-time m-ary functionality f, there exists a protocol for the computational t-secure computation of f with designated abort, for any t. Furthermore, there exists a protocol for the information-theoretic $m/2$-secure computation of f with designated abort.*

Proof: We begin by describing the protocol for computing f.

Protocol 5 (protocol for secure computation with designated abort of any functionality f):

1. Stage 1 – computation: *The parties use any protocol for secure (computational or information-theoretic) computation with* strong *designated abort in order to compute the following functionality:*[12]

   $$\overline{x} = (x_1, \ldots, x_m) \mapsto ((y_1 = f_1(\overline{x}), s_2^1, \ldots, s_m^1), s_2^2, \ldots, s_m^2)$$

   where for every i, s_i^1 and s_i^2 are binding shares *of $y_i = f_i(\overline{x})$, as by Construction* 4.

2. Stage 2 – blank round: *After the above protocol concludes, a blank-round is added as follows. If any party receives $\perp$ as output from Stage* 1*, then it sends $\perp$ to P_1 and halts outputting $\perp$.*

3. Stage 3 – outputs: *If P_1 received any $\perp$ in the blank round, then it sends $\perp$ to all parties and halts outputting $\perp$. Otherwise, for every i, it sends s_i^1 to P_i and halts, outputting y_1.*

   Party P_i outputs $\perp$ if it receives $\perp$ from P_1 (it ignores any $\perp$ it may receive from other parties). Otherwise, it checks that it received the correct reveal information from P_1. If yes, it outputs its value y_i and halts. Otherwise, it outputs $\perp$.

The security of Protocol 5 with abort, is derived from the fact that Stage 1 is run using a protocol that is secure with strong designated abort. Specifically, there are two possible cases following Stage 1: either some honest party received $\perp$ or all parties received their output. In case an honest party received $\perp$ and P_1 is honest, by Stage 2 no party will receive output. On the other hand, if no honest parties received $\perp$ and P_1 is honest, then all parties clearly receive their output (and no adversary can influence this). Finally, if P_1 is corrupted, then it can designate who should receive output and who should not. However, this is allowed by the definition and so is fine.

In order to formally prove the security of the protocol, we use the sequential composition theorem of Canetti [4]. This theorem states that we can consider a hybrid model in which an ideal call is used for Stage 1 of the protocol, whereas the other stages are as described above. It then suffices to construct a ideal-model simulator for the hybrid model in order to prove that the real and ideal models are computationally indistinguishable or statistically close (depending on whether computational or information-theoretic security is being considered). The proof of [4] is for secure computation with fair abort; however it holds also for secure computation with strong designated abort.

Let $\mathcal{A}$ be an adversary attacking Protocol 5 in the above-described hybrid model. We construct an ideal-model adversary $\mathcal{S}$ as follows, differentiating between the cases that P_1 is corrupt and P_1 is honest:

[12] We note that by Section 4, such protocols exist for any t assuming the existence of trapdoor permutations. Furthermore, for the case of $t > m/2$, no assumptions are required.

1. P_1 *is corrupt:* $\mathcal{S}$ invokes $\mathcal{A}$ and receives the inputs that $\mathcal{A}$ intends to send to the trusted party (of the hybrid model). Then, $\mathcal{S}$ forwards these to the trusted party of the ideal model unmodified. If the inputs are not valid, then in the hybrid model, all parties receive $\perp$ as output. Therefore, $\mathcal{S}$ hands $\perp$ to $\mathcal{A}$ as its output from Stage 1 and simulates all honest parties sending $\perp$ in Stage 2. $\mathcal{S}$ then halts, outputting whatever $\mathcal{A}$ does. Otherwise, if the inputs are valid, $\mathcal{S}$ receives all the corrupted parties outputs (this is the case because $\mathcal{S}$ controls P_1). $\mathcal{S}$ prepares $\mathcal{A}$'s outputs by giving it $f_1(\overline{x})$ and computing a binded sharing of each of the outputs it receives and giving them to $\mathcal{A}$. Furthermore, it gives $\mathcal{A}$ random pairs of bits for s_j^1 (for every $j \notin I$). Following this, $\mathcal{S}$ obtains the set J' from $\mathcal{A}$ instructing the hybrid-model trusted party which parties to give output to (recall $\mathcal{A}$ has strong designated abort capabilities in Stage 1 in the hybrid model); $\mathcal{S}$ records J'.

   $\mathcal{S}$ continues by simulating P_i sending $\perp$ to P_1 in the blank round, for every honest party P_i such that $i \notin J'$. Then, in the last stage, $\mathcal{A}$ (controlling P_1) sends to each honest party P_j a string ${s'}_j^1$. $\mathcal{S}$ receives these strings and defines the set of parties J to receive outputs to equal those parties in J' to whom $\mathcal{A}$ sends the same s_j^1 that $\mathcal{S}$ gave $\mathcal{A}$ in Stage 1. $\mathcal{S}$ concludes by sending J to the ideal-model trusted party and outputting whatever $\mathcal{A}$ does.

2. P_1 *is honest:* In this case, $\mathcal{S}$ begins in the same way. That is, $\mathcal{S}$ invokes $\mathcal{A}$ and receives the inputs that $\mathcal{A}$ intends to send the trusted party of the hybrid model. However, unlike in the previous case, $\mathcal{S}$ does not forward these inputs to the trusted party; rather it just records them. (If any of these inputs are invalid, then $\mathcal{S}$ internally sends $\perp$ to all corrupted parties, externally sends invalid inputs to the trusted party and halts. In the sequel, we assume that all inputs sent by $\mathcal{A}$ are valid.) Now, $\mathcal{A}$ expects to receive outputs from Stage 1 of the protocol, before it sends J' to the trusted party, designating which honest parties receive output from Stage 1. However, $\mathcal{S}$ does not have these outputs. Fortunately, $\mathcal{S}$ can perfectly simulate the corrupted parties outputs by providing them with random shares corresponding to Construction 4. That is, for each corrupted party P_i, simulator $\mathcal{S}$ generates n random triples (r_i^j, r_i^3, j) where $j \in_R \{1,2\}$ and $r_i^j, r_i^3 \in_R \{0,1\}$, for each bit of the output. Denote the share for party P_i by s_i^1. (We note that the output length is public and therefore, $\mathcal{S}$ knows how long these shares should be.) Following this, $\mathcal{S}$ obtains a set J' from $\mathcal{A}$ (this set instructs the trusted party which parties should receive output in the strong designated abort setting).

   $\mathcal{S}$ continues by simulating the rest of the protocol. Notice that honest parties only send messages to P_1 in Stage 2; since P_1 is honest, these messages need not be simulated. Thus, in this step, $\mathcal{S}$ obtains any messages sent by $\mathcal{A}$. If $\mathcal{A}$ sends $\perp$ to P_1 in Stage 2 of the protocol, then $\mathcal{S}$ sends $\perp$ to all corrupted parties (simulating messages from P_1), sends invalid inputs to the trusted party and halts. Likewise, if J' does not contain *all* the honest parties, $\mathcal{S}$ sends $\perp$ to all the corrupted parties and invalid inputs to the trusted party. (These cases correspond to the case that no parties receive their prescribed output.)

   On the other hand, if J' contains all the honest parties (i.e., no honest party received $\perp$ from Stage 1), then $\mathcal{S}$ sends the inputs that it recorded from $\mathcal{A}$ above and receives all of the corrupted parties outputs $\{f_i(\overline{x})\}_{i \in I}$. Then, for each corrupted party's output $f_i(\overline{x})$, $\mathcal{S}$ generates a share s_i^2 so that the reconstruction of s_i^1 and s_i^2 results in $f_i(\overline{x})$ (recall that this is easily achieved by Construction 4), and gives the shares to $\mathcal{A}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ does and halts.

The fact that the global output in the hybrid execution with $\mathcal{S}$ is identically distributed to the global output in a real execution with $\mathcal{A}$ is derived from the following observations. Firstly, $\mathcal{A}$'s

outputs from Stage 1 can be perfectly simulated, in both when P_1 is corrupt and when P_1 is honest. Secondly, the honest parties' messages in Stage 2 can be perfectly simulated given only the set J' sent by $\mathcal{A}$ to the hybrid-model trusted party in the ideal execution of Stage 1. Therefore, $\mathcal{A}$'s view in the hybrid-model execution is identical to its view in a real execution. It remains to show that the honest parties' outputs are also correctly distributed.

First, consider the case that P_1 is corrupt. In this case, the set of honest parties receiving output in the real model are exactly those parties P_j for whom P_1 (controlled by $\mathcal{A}$) sends the exact string s_1^j that $\mathcal{S}$ gave it in Stage 1 (and who did not receive $\perp$ from Stage 1). This is because, by the properties of Construction 4, if $\mathcal{A}$ changes any bit then it will be detected cheating except with probability 2^{-n}. Therefore, the set J sent by $\mathcal{S}$ to the trusted party contains exactly those parties who would receive output in a real execution.

Next, consider the case that P_1 is honest. In this case, all parties receive output unless P_1 sees $\perp$ in Stage 2. This can happen if $\mathcal{A}$ sends P_1 such a value, or if any honest party received $\perp$ from Stage 1. Both of these cases are detected by $\mathcal{S}$ in the hybrid-model simulation, and therefore the case that all parties receive output in the hybrid model corresponds to this case in the real model (and likewise for the case that all parties receive $\perp$). ■

References

[1] D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.

[2] D. Beaver and S. Goldwasser. Multiparty Computation with Fault Majority. In *CRYPTO'89*, Springer-Verlag (LNCS 435), 1989.

[3] M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *20th STOC*, pages 1–10, 1988.

[4] R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, Vol. 13 (1), pages 143–202, 2000.

[5] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, 2001.

[6] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO*, 2001.

[7] R. Canetti and H. Krawczyk. Universally Composable Notions of Key-Exchange and Secure Channels. In *EUROCRYPT*, 2002.

[8] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multiparty Secure Computation. In *34th STOC*, 2002.

[9] D. Chaum, C. Crepeau and I. Damgard. Multi-party Unconditionally Secure Protocols. In *20th STOC*, pages 11–19, 1988.

[10] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement. *SIAM. J. Computing*, 26(2):873–933, 1997.

[11] M. Fischer, N. Lynch, and M. Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. *Distributed Computing*, 1(1):26–39, 1986.

[12] M. Fitzi, N. Gisin, U. Maurer and O. Von Rotz. Unconditional Byzantine Agreement and Multi-Party Computation Secure Against Dishonest Minorities from Scratch. To appear in *Eurocrypt 2002*.

[13] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Byzantine Agreement Secure Against Faulty Majorities From Scratch. Manuscript 2002.

[14] Z. Galil, S. Haber and M. Yung. Cryptographic Computation: Secure Fault Tolerant Protocols and the Public Key Model. In *CRYPTO 1987*.

[15] O. Goldreich. *Secure Multi-Party Computation*. Manuscript. Preliminary version, 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[16] O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC,* pages 218–229, 1987. For details see [15].

[17] S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90*, Spring-Verlag (LNCS 537), pages 77–93, 1990.

[18] J. Kilian. Founding Cryptography on Oblivious Transfer. In *20th STOC*, pages 20–31, 1988.

[19] L. Lamport. The weak byzantine generals problem. In *JACM*, Vol. 30, pages 668–676, 1983.

[20] L. Lamport, R. Shostack, and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. and Systems*, 4(3):382–401, 1982.

[21] Y. Lindell, A. Lysyanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *34th STOC*, 2002.

[22] S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.

[23] M. Pease, R. Shostak and L. Lamport. Reaching agreement in the presence of faults. In *JACM*, Vol. 27, pages 228–234, 1980.

[24] B. Pfitzmann and M. Waidner. Information-Theoretic Pseudosignatures and Byzantine Agreement for $t >= n/3$. Technical Report RZ 2882 (#90830), IBM Research, 1996.

[25] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *21st STOC*, pages 73–85, 1989.

[26] A. Yao. How to Generate and Exchange Secrets. In *27th FOCS*, pages 162–167, 1986.

A An Overview of the Universal Composition Framework

In this appendix, we provide a brief overview of the framework of [5]; for more details, see [5]. The framework provides a formal method for defining the security of cryptographic tasks, while ensuring that security is maintained under a general composition operation in which a secure protocol for the task in question is run in a system concurrently with an unbounded number of other arbitrary protocols. This composition operation is called universal composition, and tasks that fulfill the definitions of security in this framework are called universally composable (UC).

As in other general definitions (e.g., [17, 22, 1, 4]), the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). We call the algorithm run by the trusted party an ideal functionality. Informally, a protocol securely carries out a given task if an adversary can gain no more in an attack on a real execution of the protocol, than from an attack on an ideal process where the parties merely hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction. The fact that the adversary gains no more from its attack on a real execution is formalized by saying that the result of a real execution can be *emulated* in the above ideal process. We stress that in a real execution of the protocol, no trusted party exists and the parties interact amongst themselves only.

In order to prove the universal composition theorem, the notion of emulation in this framework is considerably stronger than in previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary $\mathcal{A}$, that controls the communication channels and potentially corrupts parties. Then, security is formulated by requiring that for any adversary $\mathcal{A}$ attacking a real protocol execution, there should exist an "ideal process adversary" or simulator $\mathcal{S}$, that causes the outputs of the parties in the ideal process to be essentially the same as the outputs of the parties in a real execution. However, in the universally composable framework, an additional adversarial entity called the environment $\mathcal{Z}$ is introduced. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. (As is hinted by its name, $\mathcal{Z}$ represents the external environment that consists of arbitrary protocol executions that may be running concurrently with the given protocol.) A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. (In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$. See [5] for more motivating discussion on the role of the environment.) Note that the definition requires the "ideal-process adversary" (or simulator) $\mathcal{S}$ to interact with $\mathcal{Z}$ throughout the computation. Furthermore, $\mathcal{Z}$ cannot be "rewound".

The following *universal composition theorem* is proven in [5]: Consider a protocol π that operates in a *hybrid* model of computation where parties can communicate as usual, and in addition have ideal access to (an unbounded number of copies of) some ideal functionality $\mathcal{F}$. (This model is called the $\mathcal{F}$-hybrid model.) Furthermore, let ρ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let π^ρ be the "composed protocol". That is, π^ρ is identical to π with the exception that each interaction with the ideal functionality $\mathcal{F}$ is replaced with a call to (or an invocation of) an appropriate instance of the protocol ρ. Similarly, ρ-outputs are treated as values provided by the functionality $\mathcal{F}$. The theorem states that in such a case, π and π^ρ have essentially the same input/output behavior. Thus, ρ behaves just like the ideal functionality $\mathcal{F}$, even when composed

with an arbitrary protocol π. A special case of this theorem states that if π securely realizes some ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, then π^ρ securely realizes $\mathcal{G}$ from scratch.

B Strong Broadcast with Designated Abort

In this appendix, we prove that the broadcast with designated abort problem with the stronger validity condition, cannot be solved for $t \geq m/3$. That is,

Proposition B.1 (Proposition 3.2 – restated): *There does not exist a protocol for the* strong *broadcast with designated abort problem, for* $t \geq m/3$.

Proof: The proof of this proposition is based on the proof of Fischer et al. [11] that no Byzantine Agreement protocol can tolerate $m/3$ or more faulty parties. We prove the theorem for the special case of 3 parties; the generalization to m parties (for any m) is standard.

Assume, by contradiction, that there exists a protocol Π that solves the strong broadcast with designated abort problem for three parties A, B and C, where one may be corrupted. We designate the dealer to be B (for broadcaster). Exactly as in the proof of Fischer et al. [11], we define a hexagonal system S that intertwines two independent copies of Π. That is, let A_1, B_1, C_1 and A_2, B_2 and C_2 be independent copies of the three parties participating in Π. By independent copies, we mean that A_1 and A_2 are two instantiation of the same party A. The system S is defined by connecting party A_1 to C_2 and B_1 (rather than to C_1 and B_1); party B_1 to A_1 and C_1; party C_1 to B_1 and A_2; and so on, as in Figure 2.

Figure 2: Combining two copies of Π in a hexagonal system S.

In the system S, party B_1 has input 0 (that it is trying to broadcast); while party B_2 has input 1 (that it is trying to broadcast). Note that within S, all parties follow the instructions of Π exactly. We stress that S is *not* a "strong broadcast with designated abort" setting (where the parties are joined in a complete graph on three nodes), and therefore the definition tells us nothing directly of what the parties' outputs should be. However, S *is* a well-defined system and this implies that the parties have well-defined output distributions. The proof proceeds by showing that if Π is a correct protocol, then we arrive at a contradiction regarding the output distribution in S. We begin by showing that A_1 always outputs 0 (and not $\perp$) in S.

Claim B.2 *Party* A_1 *always halts and outputs* 0 *in the system* S.

Proof: We prove this claim by showing that there exists a corrupted party C who participates in an execution of Π and simulates the system S, with respect to A_1 and B_1's view. That is, honest parties A and B participate in an execution of "strong broadcast with designated abort", where B has input 0. In this execution, party C is corrupt and works so that A and B's views are identical to the views of A_1 and B_1 in S. The parties A, B and C work within a correct setting where there are well-defined requirements on their output distribution. Therefore, by analyzing their output in this setting, we are able to make claims regarding their output in the system S.

We begin by describing the strategy of the corrupted party C. Party C works by internally simulating some of the parties in the system S, while externally interacting with A and B. Therefore, we talk of both "internal" communication and "external" communication in our description of C. Specifically, C works by internally simulating the $\langle C_1 - A_2 - B_2 - C_2 \rangle$ part of S. That is, C internally invokes two copies of its code (C_1 and C_2) and one copy of each of A and B (for A_2 and B_2). Furthermore, the internal copy of B is given input 1 to be broadcast. Then, C begins an execution of Π with A and B, working as follows. When C externally receives a message from B, it internally forwards it to C_1. Similarly, any external messages received from A are internally forwarded to C_2. On the other hand, any message sent by internal C_1 and intended for B_1, is externally sent by C to B. Likewise, messages from C_2 to A_1 are externally sent by C to B. Finally, all messages sent between the internal (virtual) parties C_1, A_2, B_2 and C_2 are internally forwarded.

We now claim that A and B's view in the execution of Π is identical to the view of parties A_1 and B_1 in S. This can be seen as follows. First, recall that all parties in S follow the instruction of Π exactly. The only difference between an execution of S and a real execution of Π is the hexagonal form in which the parties are connected. Now, notice that C simulates this hexagonal form exactly. In particular, C simulates the $\langle C_1 - A_2 - B_2 - C_2 \rangle$ part of S perfectly. On the other hand, the $\langle A_1, B_1 \rangle$ part of S is perfectly simulated by the real parties A and B themselves ((A_1, B_1) and (A, B) have exactly the same code and input). Therefore, the views of A and B in Π, are *identical* to the views of A_1 and B_1 in S.

By the assumption that Π is a correct protocol for strong broadcast with designated abort, we have that in Π party A must output 0 (as this is the value broadcast by the honest dealer B). This holds with respect to any corrupt party C, and in particular this holds with respect to the specific party C described above. Since the views of A and B in S are identical to their views in Π, we conclude that in the system S, party A also outputs 0. This completes the proof of the claim. ■

Using analogous arguments, we obtain the following two claims:

Claim B.3 *Party C_2 always halts and outputs 1 in the system S.*

In order to prove this claim, the corrupted party is A and it works in a similar way to C in the proof of Claim B.2 above (here C_2 receives the value 1 as broadcast by B_2).

Claim B.4 *If party A_1 outputs $b \in \{0, 1\}$ in S, then C_2 outputs either b or $\perp$.*

Similarly, this claim is proven by taking the faulty party as B who follows a similar strategy to C in the proof of Claim B.2 above. However, this time the dealer is not honest. Nevertheless, the agreement property ensures that A_1 and C_2 will not output *different* non-$\perp$ values.

Combining Claims B.2, B.3 and B.4 we obtain a contradiction. This is because, on the one hand A_1 must output 0 in S (Claim B.2), and C_2 must output 1 in S (Claim B.3). On the other hand, by Claim B.4, if neither of parties A_2 and C_1 output $\perp$, then they must both output the same value. However, neither of them output $\perp$. This concludes the proof of the proposition. ■