Secure Multi-Party Computation Without Agreement*

Shafi Goldwasser
Department of Computer Science
The Weizmann Institute of Science
Rehovot 76100, Israel.
shafi@wisdom.weizmann.ac.il

Yehuda Lindell†
IBM T.J.Watson Research
19 Skyline Drive, Hawthorne
New York 10532, USA.
lindell@us.ibm.com

December 31, 2003

Abstract

It has recently been shown that authenticated Byzantine agreement, in which more than a third of the parties are corrupted, cannot be securely realized under concurrent or parallel (stateless) composition. 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 a third or more of the parties are corrupted. This is because these protocols strongly rely on the extensive use of a broadcast channel, which is in turn realized using authenticated Byzantine agreement. We remark that it was accepted folklore that the use of a broadcast channel (or authenticated Byzantine agreement) is actually essential for achieving meaningful secure multi-party computation whenever a third or more of the parties are corrupted. In this paper we show that this folklore is false. We present a mild relaxation of the definition of secure computation allowing abort. Our new definition captures all the central security issues of secure computation, including privacy, correctness and independence of inputs. However, the novelty of the definition is in *decoupling* the issue of agreement from these issues. We then show that this relaxation suffices for achieving secure computation in a point-to-point network. That is, we show that secure multi-party computation for this definition can be achieved for *any* number of corrupted parties and *without* a broadcast channel (or trusted preprocessing phase as required for running authenticated Byzantine agreement). Furthermore, this is achieved by just replacing the broadcast channel in known protocols with a very simple and efficient echo-broadcast protocol. An important corollary of our result is the ability to obtain multi-party protocols that remain secure under composition, without assuming a broadcast channel.

1 Introduction

In the setting of secure multi-party computation, a set of n parties with private inputs wish to jointly and securely compute a function of their inputs. This computation should be such that each party receives its correct output, and none of the parties learn anything beyond their prescribed output. This setting encompasses computations as simple as coin-tossing and agreement, and as complex as electronic voting, electronic auctions, electronic cash schemes, anonymous transactions, and private information retrieval schemes.

*An extended abstract of this work appeared in the 16*th DISC*, 2002.

†Most of this work was carried out while the author was at the Weizmann Institute of Science.

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.

Security in multi-party computation. A number of different definitions were proposed for secure multi-party computation (e.g., [38, 26, 6, 27, 1, 34, 9]). These definitions aimed to ensure a number of important security properties. The most central of these are:

- *Privacy:* No party should learn anything more than its prescribed output. That is, the only information that should be learned about other parties' inputs is what can be derived from the output itself.
- *Correctness:* Each party is guaranteed that the output that it receives is correct.
- *Independence of Inputs:* The corrupted parties must choose their inputs independently of the honest parties' inputs.
- *Guaranteed output delivery:* Corrupted parties should not be able to prevent honest parties from receiving their output. In other words, the adversary should not be able to carry out a denial of service attack.
- *Fairness:* Corrupted parties should receive their output if and only if honest parties do.

The standard definition today, [9] building on [27, 1, 34], formalizes the above requirements (and others) in the following general way. Consider an ideal world in which an external trusted party is willing to help the parties carry out their computation. An ideal computation takes place in this ideal world by having the parties simply send their inputs to the trusted party, who then computes the desired function and passes each party its prescribed output. Notice that all of the above security properties (and more) are ensured in such an ideal computation. The security of a real protocol is established by comparing the outcome of the protocol to the outcome of an ideal computation. Specifically, a real protocol that is run by the parties (in a world where no trusted party exists) is said to be secure, if an adversary controlling a coalition of corrupted parties can do no more harm in a real execution that in the above ideal execution. Since the adversary is unable to cause any harm in an ideal execution, this means that security is also guaranteed in a real protocol execution.

We remark that the above informal description is "overly ideal" in the following sense. It is a known fact that unless an honest majority is assumed, it is impossible to obtain generic protocols for secure multi-party computation that guarantee output delivery and fairness. The definition is therefore *relaxed* when no honest majority is assumed. In particular, under certain circumstances, honest parties may not receive their prescribed output, and fairness is not always guaranteed. In fact, a number of different levels of fairness have been considered. On one extreme, *complete fairness* states that corrupted parties receive their outputs if and only if honest parties receive their outputs (this is the notion of fairness described above and is only achievable for the case of an honest majority). On the other extreme, *no fairness* states that corrupted parties may receive their outputs even if honest parties do not. Furthermore, the decision whether or not the honest parties will receive their outputs may be made after the corrupted parties have already received their outputs. An intermediate notion that we call *partial fairness* has the following property. There exists a specified party (say P_1) such that if P_1 is honest then complete fairness is achieved. However, if P_1 is corrupt then no fairness is achieved. That is, sometimes fairness is obtained and sometimes it is not (and the fairness is thus partial).

Broadcast: In the construction of protocols, the ability to "broadcast" messages (if needed) was always assumed as a primitive, where broadcast takes on the meaning of the Byzantine Generals (or agreement) problem [31]. Namely, an honest party can deliver a message of its choice to all honest parties in a given round. Furthermore, all honest parties will receive the same message, even if the broadcasting party is corrupt. Let t be the number of corrupted parties controlled by the adversary. Then, from results obtained largely by the distributed computing community, it was known that in the setting of a *synchronous* network:

1. For $t < n/3$, Byzantine agreement can be achieved by a deterministic protocol with $O(t)$ round complexity [35], and by a randomized protocol with constant expected round complexity [19];
2. For $t \geq n/3$, broadcast is achievable using a protocol for *authenticated* Byzantine agreement, in which a public-key infrastructure for digital signatures is used [35, 31]. (This public-key infrastructure is assumed to be setup in a trusted preprocessing phase.) We note that an information theoretic analogue also exists [36]. 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 because Byzantine agreement is achievable for all values of t (with the added requirement of a trusted preprocessing phase in the case of $t \geq n/3$).

1.2 Feasibility of Secure Computation

Wide reaching feasibility results regarding secure multi-party computation were presented in the mid to late 1980's. The most central of these are as follows (recall that t equals the number of corrupted parties):

1. For $t < n/3$, secure multi-party protocols (with complete fairness and guaranteed output delivery) can be achieved for any function in a point-to-point network and without any setup assumptions. This can be achieved both in the computational setting [26] (assuming the existence of trapdoor permutations), and in the information theoretic (private channel) setting [6, 16].[1]
2. For $t < n/2$, secure multi-party protocols (with complete fairness and guaranteed output delivery) can be achieved for any function assuming that the parties have access to a broadcast channel. This can be achieved in the computational setting [26] (with the same assumptions as above), and in the information theoretic setting [37].

   Alternatively, without assuming a broadcast channel, it can be achieved in a point to point network assuming a trusted pre-processing phase for setting up a public-key infrastructure (which is then used for running authenticated Byzantine agreement).

[1] In the information theoretic setting of secure computation, the adversary is not bound to any complexity class (and in particular, is not assumed to be polynomial-time). Results in this model require no complexity or cryptographic assumptions and hold unconditionally. The only assumption used is that parties are connected via ideally private channels (i.e., it is assumed that the adversary cannot eavesdrop on the communication between honest parties).

In contrast, in the computational setting the adversary is assumed to be a probabilistic polynomial-time machine (or a polynomial-size family of circuits). Results in this model typically assume cryptographic assumptions like the existence of trapdoor permutations. We note that in this model, it is not necessary to assume that the parties have access to ideally private channels (because such channels can be implemented using encryption).

3. For $t \geq n/2$, secure multi-party protocols with partial fairness (and without guaranteed output delivery) can be achieved assuming that the parties have access to a broadcast channel and in addition assuming the existence of trapdoor permutations [38, 26, 23]. (In fact, it suffices to assume a broadcast channel and an oblivious transfer protocol [28, 29].) Some works attempting to provide higher levels of fairness have also appeared [38, 22, 27, 2].

The above results consider the static adversarial model, where the set of corrupted parties is fixed prior to the protocol execution. Analogous results also exist for the adaptive model, where the adversary can adaptively corrupt parties throughout the computation; see, for example, [3, 11].

We stress that all of the above results have been proven in the *stand-alone* model of computation where only a single execution takes place.

1.3 Byzantine Agreement and Secure Computation

As we have mentioned above, the use of a broadcast primitive in order to achieve secure computation was always viewed as non-problematic. This is due to the fact that for any number t of corrupted parties, broadcast can be securely implemented in a point-to-point network using Byzantine agreement (for $t < n/3$ standard Byzantine agreement can be used, and for $t \geq n/3$ authenticated Byzantine agreement can be used). A result of this is the fact that all known protocols for general secure computation explicitly assume broadcast, and implement it with Byzantine or authenticated Byzantine agreement.

Recent results regarding the composition of authenticated Byzantine agreement show that the use of a broadcast primitive in protocol construction may actually be very problematic. Specifically, it was shown that it is *impossible* to achieve authenticated Byzantine agreement under concurrent, or even parallel, stateless composition for $t \geq n/3$ [33]. (In the setting of stateless composition, a protocol is run many times and the honest parties run each execution independently of the others.) This puts into question the usage of authenticated Byzantine agreement in settings where protocol composition is needed. Fortunately, it is also shown in [33] that if unique session identifiers can be allocated to each execution, then security for many concurrent executions can be achieved, for any t. Thus, as long as such unique identifiers can be allocated, authenticated Byzantine agreement can still be used.

As we have mentioned, in all known protocols for general secure computation, many invocations of broadcast are used for a single execution of the secure protocol, and this broadcast is implemented using authenticated Byzantine agreement. Despite this, when considering a single execution of the secure protocol, the result of [33] poses no problem. This is due to the fact that within the secure protocol it is possible to hardwire a unique identifier to each execution of authenticated Byzantine agreement (this identifier can be the round number concatenated with the identity of the broadcasting party). In this way, it is possible to securely use authenticated Byzantine agreement. However, a problem arises if the secure protocol itself is supposed to remain secure under parallel or concurrent stateless composition. In this case, the identifiers in the different executions of authenticated Byzantine agreement will no longer be unique (e.g., the same round number and broadcasting party will be used in different executions of the protocol). It follows that in this setting, the impossibility results of [33] apply. We conclude that authenticated Byzantine agreement cannot be used as a secure implementation of broadcast within protocols that must remain secure under parallel or concurrent stateless composition.[2] However, all known protocols for secure computation *do* use authenticated Byzantine agreement. This is true even for protocols that

[2]For some applications, it may be possible to obtain unique identifiers from the high-level context. However, this may not always be the case. See [33] for more discussion on this issue and why stateless composition is preferable.

are proven secure according to the relaxed definition of secure computation that does not require guaranteed output delivery or fairness. In summary, the current state of affairs is that there are no known protocols for secure computation in a *point-to-point network* that remain secure under parallel or concurrent stateless composition, for any $t \geq n/3$ (even assuming a trusted setup phase).

1.4 Our Results

We present a further relaxation of the definition of secure multi-party computation where output delivery is not guaranteed. Our additional relaxation is very mild, yet has the effect of decoupling the issue of *agreement* from the issue of *security* in multi-party computation. Before describing our definition, we first recall the standard relaxation that is introduced for the case of $t \geq n/2$. In this case of an honest minority, guaranteed output delivery and fairness are not required. Therefore, some parties may conclude with a special abort symbol $\perp$, and not with their output. In this definition, it is required that either *all* honest parties receive their outputs or *all* honest parties output $\perp$ [23].[3] Thus the parties all *agree* on whether or not output was received. In contrast, in our new definition some honest parties may receive output while some receive $\perp$, and the requirement of agreement of abort is removed. We stress that this is the only difference between our definition and previous ones. All the other security properties (e.g., privacy, correctness and independence of inputs) are preserved.

Our main result is the construction of a protocol for secure computation according to the new definition for *any* $t < n$ and *without* a broadcast channel or setup assumption. (Our protocol assumes the same computational assumptions made, if any, by corresponding protocols that assume a broadcast channel.) We obtain this protocol (and others) by replacing the broadcast primitive in known protocols with a very simple and efficient echo-broadcast protocol. Our results hold in both the information theoretic and computational models. Unless otherwise stated, the results below refer to the static adversarial model.

A hierarchy of definitions. In order to describe our results in more detail, we present a hierarchy of definitions for secure computation. The hierarchy that we present here relates to the issues of abort (or failure to receive output) and fairness. In this work, we consider malicious adversaries, and so corrupted parties can arbitrarily deviate from the protocol specification.

1. *Secure computation without abort:* According to this definition, all parties are guaranteed to receive their output. (This is what we previously called "guaranteed output delivery".) This is the standard definition for the case of honest majority (i.e., $t < n/2$). Since all honest parties receive output, complete fairness is always obtained here.

2. *Secure computation with unanimous abort*: In this definition, it is ensured that either all honest parties receive their outputs or all honest parties abort. This definition can be considered with different levels of fairness:

   (a) *Complete fairness:* When complete fairness is achieved, the honest parties are guaranteed to receive output if the adversary does. Therefore, here one of two cases can occur: Either all parties receive output or all parties abort. This means that the adversary can conduct a denial of service attack, but nothing else. (This definition can only be achieved in the case of $t < n/2$.)

[3] We note that in private communication, Goldreich stated that the requirement in [23] of having all parties abort or all parties receive output was only made in order to simplify the definition.

(b) *Partial fairness:* As in the case of complete fairness, the adversary may disrupt the computation and cause the honest parties to abort without receiving their prescribed output. However, unlike above, the adversary may receive the corrupted parties' outputs, even if the honest parties abort (and thus the abort is not always fair). In particular, the protocol *specifies* a single party such that the following holds. If this party is honest, then complete fairness is essentially achieved (i.e., either all parties abort or all parties receive correct output). If the specified party is corrupt, then fairness may be violated. That is, the adversary receives the corrupted parties' outputs first, and then decides whether or not the honest parties all receive their correct output or all receive abort (and thus the adversary may receive output while the honest parties do not).

   Although fairness is only guaranteed in the case that the specified party is not corrupted, there are applications where this feature may be of importance. For example, in a scenario where one of the parties may be "more trusted" than others (yet not too trusted), it may be of advantage to make this party the specified party. Another setting where this can be of advantage is one where all the participating parties are trusted. However, the security problem that we are trying to protect against is that of an external party "hacking" into the machines of one of more of the parties. In such a case, it may be possible to provide additional protection to the specified party.

(c) *No fairness:* This is the same as in the case of partial fairness except that the adversary always receives the corrupted parties' outputs first. (This can be viewed as the same as partial fairness, except that the specified party is always corrupted.).

We stress that in all the above three definitions, if one honest party aborts then so do all honest parties, and so the abort is *unanimous*. This means that in the case that an abort occurs, all of the parties are aware of the fact that the protocol did not successfully terminate. This feature of having all parties succeed or fail together may be an important one in some applications.

3. *Secure computation with abort:* The only difference between this definition and the previous one is that some honest parties may receive output while others abort. That is, the requirement of unanimity with respect to abort is removed. This yields two different definitions, depending on whether partial fairness or no fairness is taken. (Complete fairness is not considered here because it only makes sense in a setting where all the parties, including the corrupted parties, either all receive output or all abort. Therefore, it is not relevant in the setting of secure computation with non-unanimous abort.)

Using the above terminology, the definition proposed by [23] for the case of $t \geq n/2$ is that of secure computation with unanimous abort and partial fairness. Our new definition is that of secure computation with abort (and partial or no fairness), and as we have mentioned, its key feature is a decoupling of the issues of secure computation and agreement (or unanimity) of abort.

Achieving secure computation with abort. In this paper, we show that secure computation with abort and partial fairness can be achieved for any $t < n$, and without a broadcast channel or a trusted pre-processing phase. We achieve this result in the following way. We begin by defining a weak variant of the Byzantine Generals problem, called *broadcast with abort,* with the following properties. First, there exists a *single* value x such that every party either outputs x or aborts. Second, when the broadcasting party is honest, the value x equals its input, similarly to the validity condition of Byzantine Generals. We call this "broadcast with abort" because as with secure computation with abort, some parties may output the correct value while other honest parties abort. We show how to achieve this type of broadcast with a simple deterministic protocol that runs in 2 rounds. Secure multi-party computation is then achieved by replacing the broadcast channel in known protocols with a broadcast with abort protocol. Despite the weak nature of agreement in this broadcast protocol, it is nevertheless enough for achieving secure multi-party computation with abort. Since our broadcast with abort protocol runs in only 2 rounds, we also obtain a very efficient transformation of protocols that work with a broadcast channel into protocols that require only a point-to-point network. In summary, we obtain the following theorem:

Theorem 1.1 (efficient transformation): *There exists an efficient protocol compiler that receives any protocol* Π *for the* broadcast model *and outputs a protocol* Π' *for the* point-to-point model *such that the following holds: If* Π *securely computes a functionality* f *with unanimous abort and with any level of fairness, then* Π' *securely computes* f *with abort and with no fairness. Furthermore, if* Π *tolerates up to* t *corruptions and runs for* R *rounds, then* Π' *tolerates up to* t *corruptions and runs for* $O(R)$ *rounds.*

Notice that in the transformation of Theorem 1.1, protocol Π' does not achieve complete fairness or partial fairness, even if Π did. Thus, fairness may be lost in the transformation. Nevertheless, meaningful secure computation is still obtained, and at virtually no additional cost.

When obtaining some level of fairness is important, Theorem 1.1 does not provide a solution. We show that partial fairness *can* be obtained without a broadcast channel for the range of $t \geq n/2$. Recall that even with a broadcast channel, complete fairness cannot be obtained in this range; therefore, we do not lose any fairness even though we work in the point-to-point model only. This result is stated in the following theorem:

Theorem 1.2 (partial fairness): *For any probabilistic polynomial-time* n*-party functionality* f*, there exists a protocol in the point-to-point model for computing* f *that is secure with abort, partially fair and tolerates any number of corruptions.*

The theorem is proven by first showing that fairness can be boosted in the point-to-point model. That is, given a generic protocol for secure multi-party computation that achieves no fairness, one can construct a generic protocol for secure multi-party computation that achieves partial fairness. (Loosely speaking, a generic protocol is one that can be used to securely compute any efficient functionality.) Applying Theorem 1.1 to known protocols for the broadcast model, we obtain secure multi-party computation that achieves no fairness. Then, using the above "fairness boosting", we obtain Theorem 1.2. We note that the round complexity of the resulting protocol is of the same order of the "best" generic protocol that works in the broadcast model. In particular, based on the protocol of [4], we obtain the first constant-round protocol in the point-to-point network for the range of $n/3 \leq t < n/2$.[4] That is:

Corollary 1.3 (constant round protocols without broadcast for $t < n/2$): *Assume that there exist public-key encryption schemes* (*or, alternatively, assume the existence of one-way functions and a model with private channels*)*. Then, for every probabilistic polynomial-time functionality* f*, there exists a* constant round *protocol in the point-to-point network for computing* f *that is secure with abort, partially fair and tolerates* $t < n/2$ *corruptions.*

[4] For the range of $t < n/3$, the broadcast channel in the protocol of [4] can be replaced by the expected constant-round Byzantine agreement protocol of [5]. However, when $n/3 \leq t < n/2$, authenticated Byzantine agreement must be used. Since there are no known (expected) constant-round protocols for authenticated Byzantine agreement, we have that the resulting protocol for secure computation is not constant-round.

Composition of secure multi-party protocols. An important corollary of our new definition is the ability to obtain secure multi-party protocols, that compose in parallel (for any $t < n$) or concurrently (for $t < n/2$), without assuming a broadcast channel. (By composition here we mean self composition, where the same protocol is run many times by the same set of parties. Furthermore, the composition is stateless, meaning that each execution is run obliviously of the others.) Until now, it was not known how to achieve such composition. This is because previously the broadcast channel in multi-party protocols was replaced by authenticated Byzantine agreement, and by [33] authenticated Byzantine agreement does *not* compose even in parallel, when $t > n/3$. (Authenticated Byzantine agreement was used because for $t > n/3$ standard Byzantine agreement cannot be applied.) Since we do not need to use authenticated Byzantine agreement to obtain secure computation, we succeed in bypassing this problem. We stress that this paper does *not* provide a comprehensive treatment of the issue of composition. Rather, our main contribution regarding this question is in removing the obstacle to obtaining composition that is due to the use of a broadcast channel in secure protocols. Seee Section 6 for more discussion on this issue.

Recently, a new framework for multi-party computation was proposed that guarantees security in a general setting where many arbitrary protocols are running concurrently [10]. Protocols that fulfill this new definition are called "universally composable". The communication model considered by [10] is one where the adversary can block messages sent by the honest parties. Notice that in this model, the adversary can always prevent the parties from receiving output. Therefore, guaranteed output delivery and fairness are not required. In this "blocking" model, universally composable multi-party computation for any $t < n$ was demonstrated, using a broadcast channel and a common reference string [14]. We show that the broadcast channel used in [14] can be replaced by a variant of our "broadcast with abort" protocol. Therefore, universally composable multi-party computation can be obtained in the point-to-point model for any $t < n$ and only assuming the existence of a common reference string and authenticated point-to-point communication.

Discussion. We propose that the basic definition of secure computation should focus on the issues of privacy, correctness and independence of inputs. In contrast, the property of agreement should be treated as an additional, and not central, feature. The benefit of taking such a position (irrespective of whether one is convinced conceptually) is that the feasibility of secure computation is completely decoupled from the feasibility of Byzantine agreement. Thus, the lower bounds relating to Byzantine agreement (and authenticated Byzantine agreement) do not imply anything regarding secure computation. Indeed, as we show, "broadcast with abort" is sufficient for secure computation. However, it lacks any flavor of agreement in the classical sense. This brings us to an important observation. Usually, proving a lower bound for a special case casts light on the difficulties in solving the general problem. However, in the case of secure computation this is not the case. Rather, the fact that the lower bounds of Byzantine agreement apply to secure computation is due to marginal issues relating to unanimity regarding the delivery of outputs, and not due to the main issues of security.

1.5 Related Work

We have recently learned of two independent and concurrent results [20, 21] studying the necessity of broadcast in secure computation, although apparently for different motivation. In [20], the question of multi-party computation in the case that the number of corruptions is $t < n/2$ is studied. 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 randomized and bypasses the $t < n/3$ lower-bound on deterministic weak Byzantine agreement protocols of [30]. They further show how this can be used in order to obtain secure computation with unanimous abort and complete fairness for the case of $t < n/2$. Thus for the range of $n/2 \leq t < n/3$, their solution achieves complete fairness whereas ours achieves only partial fairness.

Subsequently, [21] 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. They also show how their weak Byzantine agreement protocol can be used to obtain secure computation with unanimous abort and partial fairness for any $t < n$.

In comparison, we achieve secure computation with (non-unanimous) abort and partial fairness for any $t < n$. However, our focus is different. In particular, our results emphasize the fact that the issue of agreement is not central to the task of secure computation. Furthermore, removing this requirement enables us to remove the broadcast channel at almost no cost. Thus, we obtain a round-preserving transformation of secure protocols in the broadcast model to those in the point-to-point model. This is in contrast to [20, 21] who use their weak Byzantine agreement protocol in order to setup a public-key infrastructure for authenticated Byzantine agreement. They therefore incur the cost of setting up this infrastructure along with a cost of $t+1$ rounds for simulating every broadcast in the original protocol. Our protocols are therefore significantly more round efficient.[5] Finally we note that we can use the weak Byzantine agreement protocol of [21] to transform any generic r-round protocol for secure computation with abort into an $(r+t)$-round protocol with unanimous abort (and the same level of fairness). This is achieved by having the parties broadcast whether they received outputs or not after the protocol with abort concludes. It is enough to use weak Byzantine agreement for this broadcast. We therefore reduce the $O(tr)$ round complexity of [21] to $O(r+t)$, while achieving the same level of security.

Recall that [10] introduced a communication model where the adversary has control over the delivery of messages. Essentially, this definition also decouples secure computation from agreement because parties are never guaranteed to get output. In particular, the adversary is allowed to deliver output to whomever it wishes, and only these parties will ever receive output. However, the motivation of [10] is different; it aims to decouple the issues of guaranteed output delivery and fairness from the main issues of secure computation. In contrast, we focus on the question of unanimity of abort (i.e., agreement by the parties on whether or not the protocol terminated successfully).

2 Definitions – Secure Computation

In this section we present definitions for secure multi-party computation. The basic description and definitions are based on [23], which in turn follows [27, 1, 34, 9]. We actually consider a number of definitions here. In particular, we present formal definitions for secure computation with *unanimous abort* and with *abort,* with *complete fairness, partial fairness,* and *no fairness.* In addition, we refer

[5]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 an $O(r)$-round protocol that is secure with abort and partially fair (this is in contrast to the $O(tr)$ round complexity of [20, 21]). However, given a protocol that solves a specific secure computation problem, our transformation provides no fairness and does not achieve partial fairness. In order to achieve partial fairness, we must revert to a generic protocol. In contrast, the transformation of [20, 21] works for any protocol. Thus, given a very efficient protocol for a specific problem that achieves partial fairness, it may actually be "cheaper" to use the transformation of [20, 21].

to *secure computation without abort*, which 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_k the uniform distribution over $\{0,1\}^k$; for a set S we denote $s \in_R S$ when s is chosen uniformly from S; we let $[n]$ denote the set of integers $\{1,\ldots,n\}$; finally, computational indistinguishability is denoted by $\stackrel{c}{\equiv}$ and statistical closeness by $\stackrel{s}{\equiv}$. The security parameter is denoted by k.

Multi-party function evaluation. A multi-party protocol problem (for n parties $P_1,\ldots,P_n$) is cast by specifying a random process that maps vectors of inputs to vectors of outputs (one input and one output for each party). We refer to such a process as an n-ary functionality and denote it $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where $f = (f_1,\ldots,f_n)$. That is, for a vector of inputs $\overline{x} = (x_1,\ldots,x_n)$, the output-vector is a random variable $(f_1(\overline{x}),\ldots,f_n(\overline{x}))$ ranging over vectors of strings. The output for the i^{th} party (with input x_i) is defined to be $f_i(\overline{x})$.

All of the results in this paper can be generalized to *reactive functionalities* as well. In this case, the functionality to be computed is modeled by a Turing machine that continually receives inputs and generates outputs. Our definition is based on function evaluation in order to simplify the presentation. However, the actual results presented in Sections 4 and 5 hold for reactive functionalities with only very minor changes to the proofs.

Adversarial behavior. Loosely speaking, the aim of a secure multi-party 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 who may arbitrarily deviate from the protocol specification. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, parties may refuse to participate in the protocol, may substitute their local input (and enter with a different input) and may cease participating in the protocol before it terminates. Essentially, secure protocols limit the adversary to such behavior only.

Formally, the adversary is modeled by a non-uniform interactive 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 real-model adversary is deterministic.) For simplicity, most of the results in this work are proven in the *static corruption* model. Therefore, at the beginning of the execution, the adversary is given a set I of corrupted parties which it controls. That is, the adversary obtains the views of the corrupted parties, and provides them with the messages that they are to send in the execution. We note that our results can be proven in the adaptive corruption model. In particular, the protocol transformation described in Theorem 1.1 holds even for adaptive adversaries. However, our corollaries that rely on existing protocols do not all hold in the adaptive setting. Specifically, Corollary 1.3 relies on the constant-round multi-party protocol of [4] that is secure in the static model; an adaptive analogue to this protocol is not known to exist. In contrast to the above, we do consider adaptive adversaries when proving that our "broadcast with abort" protocol securely realizes the broadcast functionality under the definitions of universal composability (see Section 3.2).

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 Execution in the Ideal Model

The ideal model differs for each of the definitions. We therefore present each one separately (see Section 1.4 for an outline of the different definitions).

1. Secure computation with unanimous abort and complete fairness: According to this definition, there are two possible termination cases. In the first case, all parties (including the corrupted parties) abort without receiving output. In the second case, the protocol terminates and all parties receive their prescribed output. As we have mentioned, even assuming a broadcast channel, this definition is only achievable when the number of corrupted parties is less than $n/2$ (i.e., $|I| < n/2$). Recall that a malicious party can always substitute its input or refuse to participate. Therefore, the ideal model takes these inherent adversarial behaviors into account; i.e., by giving the adversary the ability to do this also in the ideal model. An ideal execution proceeds as follows:

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

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

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 abort symbol $\perp$.

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

Definition 1 (ideal-model computation with unanimous abort and complete fairness): *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary functionality, where $f = (f_1, \ldots, f_n)$, and let $I \subset [n]$ be the set of indices of the corrupted parties (i.e., for every $i \in I$ the adversary $\mathcal{A}$ controls P_i, and for every $j \notin I$ party P_j is honest). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$ and auxiliary input z to $\mathcal{A}$, denoted $\text{IDEAL}^{(1)}_{f,I,\mathcal{A}(z)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

2. Secure computation with unanimous abort and partial fairness: 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. Thus, we cannot prevent the "early abort" phenomenon in which the adversary receives its output, whereas the honest parties do not receive theirs (i.e., complete fairness cannot be achieved). Nevertheless, a partial notion of fairness can be achieved. That is, a party P_1 is specified so that if it is honest, then complete fairness is achieved. In contrast, if it is corrupted, then the adversary receives the corrupted parties' outputs first and then can decide whether or not the honest parties receive output or abort. We note that the abort is unanimous and thus if one honest party aborts, then so do all honest parties. The only difference from the previous definition is in the "trusted party answers remaining parties" stage. An ideal execution proceeds as follows:

Inputs: as in Definition 1.

Send inputs to trusted party: as in Definition 1.

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 outputs first). Then the corrupted P_1, depending on the views 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: as in Definition 1.

Definition 2 (ideal-model computation with unanimous abort and partial fairness): *Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary functionality, where $f = (f_1, \ldots, f_n)$, and let $I \subset [n]$ be the set of indices of the corrupted parties. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$ and auxiliary input z to $\mathcal{A}$, denoted $\text{IDEAL}^{(2)}_{f,I,\mathcal{A}(z)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

We note that this is the definition of [23] for the case of $t \geq n/2$.

3. Secure computation with unanimous abort and no fairness: This definition is very similar to the previous one, except that there is no specified party. Rather, the adversary first receives the output of the corrupted parties. Then, it decides whether all the honest parties receive output or they all abort. Formally,

Inputs: as in Definitions 1 and 2.

Send inputs to trusted party: as in Definitions 1 and 2.

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 views 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: as in Definitions 1 and 2.

Definition 3 (ideal-model computation with unanimous abort and no fairness): *Let* $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ *be an* n*-ary functionality, where* $f = (f_1, \ldots, f_n)$*, and let* $I \subset [n]$ *be the set of indices of the corrupted parties. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector* $\overline{x} = (x_1, \ldots, x_n)$ *and auxiliary input* z *to* $\mathcal{A}$*, denoted* $\text{IDEAL}^{(3)}_{f,I,\mathcal{A}(z)}(\overline{x})$*, is defined as the output vector of* $P_1, \ldots, P_n$ *and* $\mathcal{A}$ *resulting from the above described ideal process.*

The above three definitions all relate to the case of secure computation with unanimous abort. We now present the analogous definitions for the case of secure computation with abort. The only difference between the definitions is regarding the "trusted party answers remaining parties" item. In the above definitions all honest parties either receive their output or they receive $\perp$. However, here some of these parties may receive their (correct) output and some may receive $\perp$. This is where our new definition differs from past ones. We only present definitions for partial and no fairness (complete fairness only makes sense if all parties, including the adversary, either receive their outputs or $\perp$).

4. Secure computation with abort and partial fairness: As we have mentioned, the only difference between this definition and the analogous definition with unanimous abort is that 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 the corrupted P_1, depending on the views of all the corrupted parties, chooses a subset of the honest parties $J \subseteq [n] \setminus I$ 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 honest parties.

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

5. Secure computation with abort and no fairness: This definition is the same as Definition 3 with respect to the fact that the adversary always receives the output of the corrupted parties first. However, as with Definition 4, the honest parties' abort is not necessarily unanimous. That is, the adversary designates which honest parties receive their output and which receive $\perp$. We repeat the only item in which this definition differs from Definition 3:

Trusted party answers remaining parties: The adversary, depending on the views of all the corrupted parties, chooses a subset of the honest parties $J \subseteq [n] \setminus I$ 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 honest parties.

Definition 5 (ideal-model computation with abort and no fairness): *Let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ be an n-ary functionality, where $f = (f_1, \ldots, f_n)$, and let $I \subset [n]$ be the set of indices of the corrupted parties. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$ and auxiliary input z to $\mathcal{A}$, denoted $\text{IDEAL}^{(5)}_{f,I,\mathcal{A}(z)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

2.2 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). This means that the messages sent by an honest party in a given round depend on the messages that it received in previous rounds only. However, 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 are given access to a physical broadcast channel in addition to the point-to-point network). 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).[6] Furthermore, the delivery of messages between honest parties is guaranteed. Finally, we note that we do not assume any trusted preprocessing phase (that can be used to setup a public-key infrastructure, for example).[7] Most of the results in this paper relate to the above-described communication model. However, we also consider an *asynchronous* network model where the adversary has complete control over the timing and delivery of messages. (In particular, the adversary may decide to never deliver a certain message.) We stress that here the communication lines are also authenticated and private, and therefore the only thing that the adversary can do is delay or prevent a message from being sent. This network model is considered in Sections 3.2 and 6 in the context of universally composable protocols. (In this case, the ideal model is also different and is that of [10]; see Appendix A.)

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

[6] 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.

[7] One can argue that achieving authenticated and private channels in practice essentially requires a trusted preprocessing phase for setting up a public-key infrastructure. Therefore, there is no reason not to utilize this preprocessing phase in the secure multi-party 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 corrupted parties). However, we claim that the issue of achieving "secure communication channels" should be separated from the issue of achieving "secure broadcast". An example of why this is important was demonstrated in [33], who showed that authenticated Byzantine agreement does not compose (in parallel or concurrently) when 2/3 or less of the parties are honest. In contrast, secure channels can be achieved without any limitation on the protocol using them [13]; in particular, without restrictions on composability and the number of corrupted parties.

Formally,

Definition 6 (real-model execution): *Let f be an n-ary functionality and let Π be a multi-party protocol for computing f. Furthermore, let $I \subset [n]$ be the set of indices of the corrupted parties. Then, the* joint execution of Π under $(\mathcal{A}, I)$ in the real model *on input vector $\overline{x} = (x_1, \ldots, x_n)$ and auxiliary input z to $\mathcal{A}$, denoted $\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the protocol interaction, where for every $i \in I$, 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.*

A remark on synchronous versus asynchronous networks. As we have mentioned above, most of the results of this paper are presented in the synchronous network model. However, this is for the sake of simplicity only. In fact, when output delivery is *not* guaranteed, asynchronous communication does not present any additional hardship. Specifically, the honest parties should merely wait until receiving all round i messages before sending their message for round $i+1$. This ensures that the only difference between a synchronous and an asynchronous execution is the ability of the adversary to prevent the parties from receiving output. When output delivery is not guaranteed, this is anyway allowed.

2.3 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 the adversary can do no more harm in a real protocol execution that in the ideal model (where security trivially holds). This is formulated by saying that adversaries in the ideal model are able to simulate adversaries in an execution of a 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. In the second, we consider unbounded adversaries and require that the simulation be such that the output distributions of the two executions are statistically close.

Definition 7 (computational security): *Let f and Π be as above. We say that protocol Π is a protocol for the* computational t-secure computation of f with unanimous abort (*resp.,* with abort) *and with* complete fairness (*resp., with* partial fairness *or with* no fairness)*, if for every non-uniform polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic (expected) polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [n]$ with $|I| < t$,*

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

where the value of $\alpha \in \{1,2,3,4,5\}$ depends on whether secure computation with unanimous abort or with abort is being considered, and whether complete fairness, partial fairness or no fairness is required.

Definition 8 (information-theoretic security): *Let f and Π be as above. We say that protocol Π is a protocol for the* information-theoretic t-secure computation of f with unanimous abort (*resp.,* with abort) *and with* complete fairness (*resp., with* partial fairness *or with* no fairness)*, if for every non-uniform adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic adversary $\mathcal{S}$ for the ideal model such that for every $I \subset [n]$ with $|I| < t$,*

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

where the value of $\alpha \in \{1, 2, 3, 4, 5\}$ depends on whether secure computation with unanimous abort or with abort is being considered, and whether complete fairness, partial fairness or no fairness is required.

3 Broadcast with Abort

In this section, we present a weak variant of the Byzantine Generals problem, that we call *"broadcast with abort"*. The main idea is to weaken both the agreement and validity requirements so that some parties may output the broadcast value x while others output $\perp$. This primitive can be defined in line with definitions presented in the previous section as follows. Let f be the functionality defined by $f_i(\overline{x}) = x_1$ for all i. That is, $f(x_1, \ldots, x_n) = (x_1, \ldots, x_1)$. This is clearly a broadcast functionality, where the first party broadcasts its value to all others (and all other inputs are ignored). Now, a protocol solves the broadcast with abort problem if it securely computes f *with abort and no fairness.* This definition is very different to the direct (non simulation-based) definitions usually provided for Byzantine agreement protocols. We therefore prove the security of our protocol under the following definition (that is not according to the definitions in the previous section, but is more in line with the standard definitions of problems of this kind):

Definition 9 (broadcast with abort): *Let $P_1, \ldots, P_n$, be n parties and let P_1 be the dealer with input x. In addition, let $\mathcal{A}$ be an adversary who controls up to t of the parties (which may include P_1). A protocol solves the* broadcast with abort *problem, tolerating t corruptions, if for any adversary $\mathcal{A}$ as above, 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.) We now present a simple protocol that solves the broadcast with abort problem for *any* t. As we will see later, despite its simplicity, this protocol suffices for obtaining secure computation with abort.

Protocol 1 (broadcast with 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 from P_1 in the previous round. If P_i did not receive a value from P_i in the first round, then it sets $x^i = \perp$.*

     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 for every x^i_j that P_i receives, it holds that $x^i_j = x^i$). Otherwise, it 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 abort problem, and tolerates* any $t < n$ *corruptions.*

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 $x^j = x'$. Furthermore, 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 $\perp$.

■

3.1 Strengthening Broadcast with Abort

A natural question to ask is whether or not we can strengthen Definition 9 in one of the following two ways (and still obtain a protocol that tolerates $t \geq n/3$ corruptions):

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

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

It is easy to see that the 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 do not exist deterministic protocols for the case of $t \geq n/3$ [30]. For what can be done if one utilizes randomized protocols, see Section 1.5. Regarding the strengthening of the *validity requirement,* the resulting definition implies a problem known as "Crusader Agreement" (outputting $\perp$ can be interpreted as "explicitly" knowing that the dealer is corrupted). This was shown to be unachievable for any $t \geq n/3$ in [17] (this lower bound holds for randomized protocols as well). We therefore conclude that the "broadcast with abort" requirements cannot be strengthened in either of the above two ways (for deterministic protocols), without incurring a $t < n/3$ lower bound.

3.2 Universally Composable Broadcast

In this section, we consider the problem of achieving secure broadcast in the framework of universal composability. A very brief overview of universal composability can be found in Appendix A; see [9] for more details.

The model that we consider here differs from that described in Section 2 in a number of ways. First, we consider a model of *adaptive* rather than static corruptions. Second, we consider an *asynchronous* network setting with ideally authenticated communication lines. In such a setting, the adversary cannot modify messages sent by the honest parties, but does have control over the timing and delivery of these messages. We note that the adversary's control is complete, and thus it may decide to never deliver certain messages. This is modeled in the following way: The parties send messages by placing them in an "out-box", and the adversary then delivers (or does not deliver) them at will. This capability is given to the adversary in both the real and ideal models (and thus the adversary also delivers – or does not deliver – messages between the honest parties and the trusted third party).[8] This is very similar to our definition of secure computation with abort and no fairness. In particular, output delivery is not guaranteed. Furthermore, the adversary can choose to deliver outputs to any subset of the parties that it wishes to (and can decide after having seen the corrupted parties' outputs). Thus, there is a fundamental difference between the notion of universally composable broadcast in this model and all the other works on Byzantine agreement in the asynchronous model where output is guaranteed [8, 15]. (As we have discussed at length in the introduction, this much weaker notion of broadcast nevertheless suffices for the purposes of secure computation with abort.)

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, for any $t < n$. Essentially this means that any protocol that uses a broadcast channel can be implemented in the point-to-point model while achieving the same effect. An important corollary of this result is the existence of universally composable multi-party computation for $t \geq n/3$, in a point-to-point network. This corollary is obtained by applying our secure realization of the broadcast functionality with known universally composable multi-party protocols; see Section 6.

The ideal broadcast functionality is defined in Figure 1.

Broadcast Functionality

The broadcast functionality is defined as follows, running with parties $P_1, \ldots, P_n$ and an ideal-model adversary $\mathcal{S}$:

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

Figure 1: The ideal broadcast functionality

We now present our protocol for securely realizing universally composable broadcast. This protocol is a slightly modified version of Protocol 1 and is suited for an asynchronous setting:

Protocol 2 (universally composable broadcast):

- **Input:** *P_i has input* $(\mathsf{broadcast}, x)$.
- **The Protocol:**
  1. *P_i sends x to all parties.*
  2. *Upon receiving a value x^j from P_i, party P_j sends the value x^j that it received to all other parties.*
  3. *Party P_j waits to receive a message from every party (other than party P_i). Denote the message received from P_k by x^j_k. Then, P_j outputs* $(\mathsf{broadcast}, P_i, x^j)$ *if and only if for every k, it holds that $x^j_k = x^j$ (i.e., $x^j = x^j_1 = \cdots = x^j_n$). Otherwise, it outputs nothing.*

[8] In some versions of the UC framework, the adversary does not deliver the messages between the trusted party and the honest parties in the ideal model; rather, these messages are sent directly and immediately. In such a case, the adversary's ability to influence when (if at all) an honest party receives its output is then explicitly defined in the functionality. This technical issue is of little relevance to our protocol here.

The main result of this section is the following proposition:

Proposition 3.2 *Protocol* 2 *is a universally composable protocol for securely realizing the ideal broadcast functionality in the presence of malicious, adaptive adversaries* (*and in an asynchronous network*)*.*

Proof: Let $\mathcal{A}$ be a real-model adversary attacking Protocol 2. We construct an ideal model adversary $\mathcal{S}$ that internally incorporates $\mathcal{A}$ and interacts with the trusted party in an ideal execution. $\mathcal{S}$ forwards all messages unmodified between $\mathcal{A}$ and the environment $\mathcal{Z}$. In addition, $\mathcal{S}$ simulates a real execution for $\mathcal{A}$. We separately describe the simulation for the case that the dealer is corrupted before it sends its messages in the first round, and the case that it is honest (recall that P_i is the dealer in the execution):

- *Case* 1 – P_i *is corrupt:* In the first round, $\mathcal{A}$ (controlling P_i) sends messages to some, or possibly all, of the honest parties. $\mathcal{S}$ simulates the response of the honest parties to these messages and waits until all the first-round messages of the protocol have been delivered by $\mathcal{A}$ in the simulation. Once these messages have been delivered, $\mathcal{S}$ works as follows. If $\mathcal{A}$ sent the same first-round message x to *all* the honest parties, then $\mathcal{S}$ (controlling P_i) sends $(\mathsf{broadcast}, x)$ to the trusted party in the ideal execution. Following this, $\mathcal{S}$ continues in the internal simulation for $\mathcal{A}$ of the honest parties' messages until the protocol terminates. In addition, $\mathcal{S}$ delivers output from the trusted party to a given honest party, whenever that honest party receives all of its second-round messages, and all of these messages equal x.

  If $\mathcal{A}$ did not send the same first-round message to all the honest parties, then $\mathcal{S}$ internally simulates for $\mathcal{A}$ all the messages that the honest parties would send, but does not send anything to the trusted party.

- *Case* 2 – P_i *is honest:* $\mathcal{S}$ receives $(\mathsf{broadcast}, P_i, x)$ from the trusted party and simulates a real execution for $\mathcal{A}$ where P_i is the dealer. First, $\mathcal{S}$ simulates P_i sending x to all the parties and the honest parties responses to these messages. Then, $\mathcal{S}$ receives back the messages sent by $\mathcal{A}$ to the honest parties. As in the previous case, $\mathcal{S}$ delivers output from the trusted party to an honest party, whenever the honest party receives all of its second-round messages, and all of these messages equal x.

It remains to describe $\mathcal{S}$'s actions for corruptions. This is straightforward here because honest parties hold no private information whatsoever (the protocol is deterministic and so they don't even have random tapes). Therefore, when $\mathcal{A}$ corrupts a party, it learns nothing beyond what was known before. Thus, $\mathcal{S}$ need not provide any information to $\mathcal{A}$ and can just corrupt the appropriate ideal-process party.

We claim that the output of the environment $\mathcal{Z}$ in an ideal execution with $\mathcal{S}$ is *identically distributed* to its output in a real execution with $\mathcal{A}$. In order to see this, first notice that the simulator $\mathcal{S}$ is deterministic. Therefore, the only random coins used are on $\mathcal{Z}$'s random-tape (recall that we can always assume that $\mathcal{A}$ is deterministic). Given this observation, we make the following strong claim. Let r_z denote a series of coins and let $\mathcal{Z}_{r_z}$ denote $\mathcal{Z}$ when its random tape contains r_z. Then, the output of the honest parties in a real execution with $\mathcal{A}$ and $\mathcal{Z}_{r_z}$ *equals* the output of the honest parties in an ideal execution with $\mathcal{S}$ and $\mathcal{Z}_{r_z}$. Furthermore, the series of messages received by $\mathcal{A}$ from the honest parties in a real execution with $\mathcal{A}$ and $\mathcal{Z}_{r_z}$ *equals* the series of messages received by $\mathcal{A}$ from $\mathcal{S}$ in the simulation of an ideal execution of $\mathcal{S}$ with $\mathcal{Z}_{r_z}$. This suffices because $\mathcal{Z}$'s view consists of the honest parties' outputs and interaction with $\mathcal{A}$. Since $\mathcal{A}$'s view and the honest parties' outputs are the same in the real and ideal executions, it follows that $\mathcal{Z}$'s output is also the same. (This holds for all possible random tapes, and thus it also holds for a randomly chosen tape.)

We first show that for every $r_z \in \{0,1\}^*$, an honest party outputs x in a real execution with $\mathcal{A}$ and $\mathcal{Z}_{r_z}$ if and only if it outputs x in an ideal execution with $\mathcal{S}$ and $\mathcal{Z}_{r_z}$. First, consider the case that the dealer P_i is honest when it sends its first-round messages. In a real execution, an honest party outputs x if it received x in the first round and receives x from all other parties in the second round. Since P_i is honest, all honest parties indeed receive P_i's input x in the first round. Furthermore, $\mathcal{S}$ only delivers output from the trusted party to an honest party if this honest party receives all the second-round messages and they are all x, exactly as in the protocol. Therefore, the output of an honest party in a real and ideal execution is identical when P_i is honest. Next, consider the case that P_i is corrupt when it sends its first-round messages. If P_i, controlled by $\mathcal{A}$, sends all the honest parties the same value x in the first round, then this is exactly the same as the case that P_i is honest. (Note that if only one party is honest, then "all honest parties" are always sent the same value.) However, if P_i does not act in this way, then $\mathcal{S}$ sends nothing to the trusted party. Thus, it remains to show that in a real execution with such a P_i, no honest party would output a value. If P_i sends two different messages to two honest parties in the first round (i.e., if there exist j and k such that $x^j \neq x^k$), then by the protocol definition, all honest parties who output a value see both x^j and x^k. (Otherwise, they did not receive all second-round messages and so do not yet output anything.) Therefore, no honest party concludes with output. Thus, the output of an honest party in a real and ideal execution is also identical in the case that P_i is corrupt.

It remains to show that the messages received by $\mathcal{A}$ from the honest parties in a real execution with $\mathcal{Z}_{r_z}$ are identical to the messages received by $\mathcal{A}$ from $\mathcal{S}$ in an ideal execution with $\mathcal{Z}_{r_z}$. However, this follows directly from the description of $\mathcal{S}$, who always simulates the exact response of the honest parties to the messages sent by $\mathcal{A}$. This completes the proof. ■

In Section 6, we show how Proposition 3.2 is used to obtain universally composable protocols in a point-to-point network.

4 Secure Computation with Abort and No Fairness

In this section, we show that any protocol for secure computation (with unanimous abort and any level of fairness) that uses a broadcast channel can be "compiled" into a protocol for the point-to-point network that achieves secure computation with abort and no fairness. Furthermore, the fault tolerance of the compiled protocol is the same as the original one. Actually, we assume that the protocol is such that all parties terminate in the same round. We say that such a protocol has *simultaneous termination*. Without loss of generality, we also assume that all parties generate their output in the last round.

Before formally stating this result, we note that our compilation can actually be applied to *any* protocol that uses a broadcast channel (and not only secure protocols). The result of our compilation on a general protocol would be as follows: let J be the indices of the parties in the compiled protocol who *do not abort*. Then, the outputs of the non-aborting parties $\{P_j\}_{j\in J}$ are the same in the compiled protocol (that is run in a point-to-point network) and the original protocol (that is run with a broadcast channel). We now state the result of this section, with the focus on secure protocols:

Theorem 1 *There exists a (polynomial-time) protocol compiler that takes any protocol* Π *(with simultaneous termination) for the* broadcast model*, and outputs a protocol* Π' *for the* point-to-point model *such that the following holds: If* Π *is a protocol for the information-theoretic (resp., computational)* t*-secure computation of a functionality* f *with unanimous abort and any level of fairness, then* Π' *is a protocol for the information-theoretic (resp., computational)* t*-secure computation of* f *with abort and no fairness.*

Combining Theorem 1 with known protocols with simultaneous termination (specifically, [37] and [26], we obtain the following corollaries:

Corollary 2 (information-theoretic security – compilation of [37]): *For every probabilistic polynomial-time* n*-ary functionality* f*, there exists a protocol in the point-to-point model, for the information-theoretic* $n/2$*-secure computation of* f *with abort and no fairness.*

Corollary 3 (computational security – compilation of [26]): *For every probabilistic polynomial-time* n*-ary functionality* f*, there exists a protocol in the point-to-point model, for the computational* t*-secure computation of* f *with abort and no fairness, for* any t*.*

We now proceed to prove Theorem 1.

Proof of Theorem 1: 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. This simulation is not carried out using Byzantine Generals because this is not possible for $t \geq n/3$. Rather, it is simulated using a protocol for "broadcast with abort", which as we have seen can be achieved for any t. 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 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.

Components of the compiler:

1. Broadcast with abort executions: Each broadcast of the original protocol (using the assumed broadcast channel) is replaced with an execution of the broadcast with abort protocol.

2. Blank rounds: Following each broadcast with abort execution, a blank round is added in which no protocol messages are sent. Rather, these blank rounds are used to enable the parties to notify each other in the case that they abort. Specifically, if a party receives $\perp$ in a broadcast with abort execution, then it sends $\perp$ to all parties in the blank round that immediately follows and halts. Likewise, if a party receives $\perp$ in a blank round, then it sends $\perp$ to all parties in the next blank round and halts.

Thus each round of the original protocol is transformed into 3 rounds in the compiled protocol (2 rounds for broadcast with abort and an additional blank round). We now formally define the protocol compiler:

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

- *The parties use* broadcast with abort *in order to emulate each broadcast message of protocol* Π. *Each round of* Π *is expanded into* 3 *rounds in* Π'*:* broadcast with abort *is run in the first* 2 *rounds, and the third round is a* blank round. *Point-to-point messages of* Π *are sent unmodified in* Π'. *More exactly, the parties emulate* Π *according to the following instructions (for simplicity, we count the first round of* Π *to be round* 0*):*
  1. Broadcasting messages: *Let* P_i *be a party who is supposed to broadcast a message* m *in the* j^{th} *round of* Π. *Then, in the* j^{th} *broadcast simulation of* Π' (*i.e., in rounds* $3j$ *and* $3j+1$ *of* Π'), *all parties run an execution of* broadcast with abort *in which* P_i *plays the dealer role and sends* m.
  2. Sending point-to-point messages: *Any message that* P_i *is supposed to send to* P_j *over the point-to-point network in the* j^{th} *round of* Π *is sent by* P_i *to* P_j *over the point-to-point network in round* $3j$ *of* Π'.
  3. 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* broadcast with abort *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* (*i.e., in the blank round following the execution of* broadcast with abort), *and halts immediately thereafter.*
  4. 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. In the* 2 *rounds preceding the next blank round, Party* P_i *does* not *send any point-to-point messages or messages belonging to a broadcast execution.* (*We note that if this blank round is the last round of the execution, then* P_i *simply halts.*)
  5. Output: *If a party* P_i *received* $\perp$ *at any point in the execution* (*in an execution of* broadcast with abort *or in a blank round*), *then it outputs* $\perp$. *Otherwise, it outputs the value specified by* Π.

By the assumption in the theorem, Π is t-secure with unanimous abort (and any level of fairness). In order to prove that the compiled protocol Π' is t-secure with abort and no fairness, 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 the adversary with the additional ability of prematurely halting honest parties. We now define the hybrid protocol $\tilde{\Pi}$ and show that it is t-secure with abort and no fairness:

Lemma 4.1 *Let* Π *be a protocol in the broadcast model for the information-theoretic* (*resp., computational*) t*-secure computation of a functionality* f *with unanimous abort and any level of fairness. Then, define protocol* $\tilde{\Pi}$ (*also for the broadcast model*) *as follows:*

1. *Following each round of* Π, *add a blank round.*
2. *If in a blank round,* P_i *receives a* $\perp$ *message, then* P_i *sends* $\perp$ *to* P_j *for all* $j \neq i$ *in the next blank round and halts.* P_i *also does not broadcast any message or send any point-to-point messages in the next round of* Π (*before the blank round where it sends all the* $\perp$ *messages*).
3. *Apart from the above, the parties follow the instructions of* Π.
4. Output: *If a party* P_i *received* $\perp$ *in any blank round, then it outputs* $\perp$. *Otherwise, it outputs the value specified by* Π.

Then, $\tilde{\Pi}$ *is a protocol for the information-theoretic (resp., computational)* t*-secure computation of* f *with abort and no fairness.*

Proof: We prove this theorem for the case that Π is computationally t-secure with unanimous abort and partial fairness. The other cases (information theoretic security and security with complete fairness or no fairness) are proved in a similar way. Let $\tilde{\mathcal{A}}$ be a real-model adversary attacking $\tilde{\Pi}$. Our aim is to construct an ideal-model simulator $\tilde{\mathcal{S}}$ for $\tilde{\mathcal{A}}$. In order to do this, we must use the fact that for any adversary $\mathcal{A}$ attacking protocol Π, there exists an ideal-model simulator $\mathcal{S}$. Unfortunately we cannot apply $\mathcal{S}$ to $\tilde{\mathcal{A}}$ because $\mathcal{S}$ is a simulator for protocol Π and $\tilde{\mathcal{A}}$ participates in protocol $\tilde{\Pi}$. We therefore first construct an adversary $\mathcal{A}$ that attacks Π from the adversary $\tilde{\mathcal{A}}$ that attacks $\tilde{\Pi}$. The construction of $\mathcal{A}$ is such that the output distribution of an execution of Π with $\mathcal{A}$ is very similar to the output distribution of an execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$. Having constructed $\mathcal{A}$, it is then possible to apply the simulator $\mathcal{S}$ that we know is guaranteed to exist. We therefore obtain a simulator $\tilde{\mathcal{S}}$ for $\tilde{\mathcal{A}}$ by first "transforming" $\tilde{\mathcal{A}}$ into $\mathcal{A}$ and then applying $\mathcal{S}$. As we will show, the resulting simulator $\tilde{\mathcal{S}}$ is as required for $\tilde{\Pi}$. Details follow.

We begin by defining the adversary $\mathcal{A}$ who attacks Π. Adversary $\mathcal{A}$ internally incorporates $\tilde{\mathcal{A}}$ and has internal communication with $\tilde{\mathcal{A}}$ and external communication with the honest parties executing protocol Π.

Adversary $\mathcal{A}$ for Π:

- **Input:** $\mathcal{A}$ receives an auxiliary input z, an input sequence $\{x_i\}_{i\in I}$ and a series of random-tapes $\{r_i\}_{i\in I}$. (Recall that $\mathcal{A}$ controls all the parties in the set I. Thus, corrupted party P_i's input and random-tape equal x_i and r_i, respectively.)
- **Execution:**
  1. *Invoke $\tilde{\mathcal{A}}$:* $\mathcal{A}$ begins by invoking $\tilde{\mathcal{A}}$ upon auxiliary input z, input sequence $\{x_i\}_{i\in I}$ and random-tapes $\{r_i\}_{i\in I}$.
  2. *Emulation before $\tilde{\mathcal{A}}$ sends any $\perp$ messages:* $\mathcal{A}$ internally passes to $\tilde{\mathcal{A}}$ all the messages that it externally receives from the honest parties (through broadcast or point-to-point communication). Likewise, $\mathcal{A}$ externally broadcasts in Π any message that $\tilde{\mathcal{A}}$ broadcasts in $\tilde{\Pi}$, and $\mathcal{A}$ externally sends P_j in Π any message that $\tilde{\mathcal{A}}$ sends P_j in $\tilde{\Pi}$.
  3. *Emulation after $\tilde{\mathcal{A}}$ sends a $\perp$ message:* Once $\tilde{\mathcal{A}}$ sends a $\perp$ message in an execution of $\tilde{\Pi}$, the honest parties in Π and $\tilde{\Pi}$ may behave differently (in particular, a party receiving $\perp$ may continue to send messages in Π, whereas it would halt in $\tilde{\Pi}$). Therefore, $\mathcal{A}$ filters messages sent by honest parties in Π so that $\tilde{\mathcal{A}}$'s view equals what it would in an execution of $\tilde{\Pi}$. That is:

     In the round of Π following the first $\perp$ message sent by $\tilde{\mathcal{A}}$, adversary $\mathcal{A}$ forwards to $\tilde{\mathcal{A}}$ only the point-to-point and broadcast messages sent by a party P_j who did not receive $\perp$ from $\tilde{\mathcal{A}}$ in the previous (simulated) blank round of $\tilde{\Pi}$. Furthermore, $\mathcal{A}$ simulates for $\tilde{\mathcal{A}}$ the sending of all the $\perp$ messages that would be sent in the next blank round of $\tilde{\Pi}$ (if one exists), and halts.
  4. *Output:* $\mathcal{A}$ outputs whatever $\tilde{\mathcal{A}}$ does.

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 claim that the joint distribution of the outputs of all parties not outputting $\perp$ and the adversary is identical in Π and $\tilde{\Pi}$. We begin with some notation:

- Let $\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x},\overline{r})$ be the global output of an execution of Π with adversary $\mathcal{A}$ with auxiliary input z, inputs $\overline{x}$, and random-tapes $\overline{r}$ (i.e., $\overline{r}=(r_1,\ldots,r_n)$ where P_i receives random-tape r_i). (Thus, $\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x}) = \{\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x},U_{|\overline{r}|})\}$.)
- For any subset $J \subseteq [n]$, denote by $\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x},\overline{r})|_J$, the *restriction* of $\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x},\overline{r})$ to the outputs of $\mathcal{A}$ and all parties P_j for $j \in J$. We stress that $\mathcal{A}$'s output is included in this restriction.
- In any execution of $\tilde{\Pi}$, it is possible to divide the parties into those who output $\perp$ and those who do not output $\perp$. We note that the set of parties outputting $\perp$ is chosen by the adversary $\tilde{\mathcal{A}}$ and is dependent solely on the auxiliary input z and the parties' inputs $\overline{x}$ and random-tapes $\overline{r}$. We denote by $J = J_{\tilde{\mathcal{A}}(z)}(\overline{x},\overline{r})$ the set of parties who do not output $\perp$ in an execution of $\tilde{\Pi}$ with adversary $\tilde{\mathcal{A}}$ (and where the auxiliary input is z and the parties' inputs and random-tapes are $\overline{x}$ and $\overline{r}$). (Notice that $J_{\tilde{\mathcal{A}}}$ is a deterministic function depending only on z, $\overline{x}$ and $\overline{r}$. This is because, without loss of generality, the adversary $\tilde{\mathcal{A}}$ is deterministic.) We also denote by $J_{\tilde{\mathcal{A}}(z)}(\overline{x})$ a random variable taking values over $J_{\tilde{\mathcal{A}}(z)}(\overline{x},\overline{r})$ for uniformly distributed $\overline{r}$.

We now consider the joint distribution of the outputs of the adversary and the parties in $J_{\tilde{\mathcal{A}}(z)}(\overline{x},\overline{r})$ (i.e., those not outputting $\perp$). We claim that these distributions are identical in Π with $\mathcal{A}$ and in $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$. Using the above notation, we claim that for every adversary $\tilde{\mathcal{A}}$ and set of corrupted parties I, for all auxiliary inputs z, and for all input and random-tape sequences $\overline{x}$ and $\overline{r}$,

$$\text{REAL}_{\Pi,I,\mathcal{A}(z)}(\overline{x},\overline{r})|_{J_{\tilde{\mathcal{A}}(z)}(\overline{x},\overline{r})} = \text{REAL}_{\tilde{\Pi},I,\tilde{\mathcal{A}}(z)}(\overline{x},\overline{r})|_{J_{\tilde{\mathcal{A}}(z)}(\overline{x},\overline{r})} \tag{1}$$

where $\mathcal{A}$ is as defined above. We now prove Eq. (1). First, it is clear that $\tilde{\mathcal{A}}$'s view in a real execution of $\tilde{\Pi}$ is identical to its view in the simulation by $\mathcal{A}$. Therefore, $\mathcal{A}$'s output in Π equals $\tilde{\mathcal{A}}$'s output in $\tilde{\Pi}$. Next, notice that if there exists an honest party that does not output $\perp$ in $\tilde{\Pi}$, then it must be that $\perp$ messages were sent in the last blank round only. However, this means that any honest party not receiving such a message has an identical view in Π and $\tilde{\Pi}$. Therefore, the outputs of all such parties are identical in Π and $\tilde{\Pi}$. Eq. (1) follows. We stress that the parties who output $\perp$ in $\tilde{\Pi}$ may have very different outputs in Π (and in particular may output their prescribed outputs). Nevertheless, at this stage we are only interested in those parties not outputting $\perp$.

We now proceed to construct a simulator $\tilde{\mathcal{S}}$ for $\tilde{\mathcal{A}}$. Intuitively, $\tilde{\mathcal{S}}$ works by using the simulator $\mathcal{S}$ for $\mathcal{A}$, where $\mathcal{A}$ is derived from $\tilde{\mathcal{A}}$ as above. Recall that $\mathcal{A}$ is an adversary for the secure protocol Π, and thus a simulator $\mathcal{S}$ is guaranteed to exist for $\mathcal{A}$.

Simulator $\tilde{\mathcal{S}}$: First consider an adversary $\mathcal{A}$ for Π constructed from the adversary $\tilde{\mathcal{A}}$ as described above. By the security requirements of Π, for every adversary $\mathcal{A}$ there exists an ideal-model simulator $\mathcal{S}$ for $\mathcal{A}$. Simulator $\tilde{\mathcal{S}}$ for $\tilde{\mathcal{A}}$ works by emulating an ideal execution for this $\mathcal{S}$. This involves emulating the interaction that $\mathcal{S}$ has with the trusted third party for its ideal model (while $\tilde{\mathcal{S}}$ interacts with the trusted party in its own ideal model). Recall that $\tilde{\mathcal{S}}$ works in an ideal model for secure computation with abort and no fairness (i.e., according to Definition 5), whereas $\mathcal{S}$ works in an ideal model for secure computation with unanimous abort and partial fairness (i.e., according to Definition 2). Thus, essentially $\tilde{\mathcal{S}}$ has "more power" than $\mathcal{S}$. Also, recall that where partial fairness holds, there are two cases depending on whether or not P_1 is corrupted. If P_1 is not corrupted, then essentially all parties receive output at the same time. If P_1 is corrupted, then the adversary receives the corrupted parties' outputs first and then decides whether the honest parties all receive output or all abort.

We now describe the simulator. Let $\tilde{\mathcal{S}}$'s auxiliary input be z and its input be a set of values $\{x_i\}_{i\in I}$. Furthermore, let r be the contents of $\tilde{\mathcal{S}}$'s random tape. Then, $\tilde{\mathcal{S}}$ sets $\mathcal{S}$'s random tape to r and internally invokes $\mathcal{S}$ upon auxiliary input z and input series $\{x_i\}_{i\in I}$. Next, $\tilde{\mathcal{S}}$ works as an intermediary between $\mathcal{S}$ and the trusted party. That is, $\tilde{\mathcal{S}}$ obtains the input values $\{x'_i\}_{i\in I}$ sent by $\mathcal{S}$ and externally sends these same values to the trusted party. Once $\tilde{\mathcal{S}}$ forwards these inputs to the trusted party, it receives back all the corrupted parties' outputs (recall that $\tilde{\mathcal{S}}$ interacts in an ideal model with no fairness). $\tilde{\mathcal{S}}$ then forwards these outputs to $\mathcal{S}$. We distinguish between two cases regarding the way the simulator $\mathcal{S}$ must work:

1. P_1 *is not corrupted:* in this case, $\mathcal{S}$ concludes its execution at this point, outputting some value. (Recall that $\mathcal{S}$ works in an ideal model of partial fairness. Therefore, when P_1 is not corrupted, $\mathcal{S}$ sends no instructions to the trusted party after sending its inputs.)

2. P_1 *is corrupted:* in this case, $\mathcal{S}$ first instructs the trusted party to either send all the honest parties their outputs or send them all $\perp$. $\mathcal{S}$ then concludes, outputting some value.

$\tilde{\mathcal{S}}$ ignores the instruction sent to the trusted party in the second case, and sets its output to be whatever $\mathcal{S}$ output. It remains to define the set J that $\tilde{\mathcal{S}}$ sends to the trusted party in the "trusted party answers remaining parties" stage of its ideal execution (recall that all honest parties in J 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. In particular, if $\tilde{\mathcal{A}}$ sent a $\perp$ message before the last blank round, then all honest parties abort (and $J=\phi$). Otherwise, all parties receive output except for those receiving $\perp$ messages in the last blank round. Therefore, $\tilde{\mathcal{S}}$ examines this view and defines the set J accordingly.[9] Once J 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 abort and no fairness with adversary $\tilde{\mathcal{S}}$ 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) in the ideal model. That is, we show that the outputs of parties not outputting $\perp$ in an execution of $\tilde{\Pi}$ are the same in an ideal execution (by Def. 2) with $\mathcal{S}$ and in an ideal execution (by Def. 5) with $\tilde{\mathcal{S}}$. Before claiming this formally, we introduce the following notation. Let $\text{IDEAL}^{(\alpha)}_{f,I,\tilde{\mathcal{S}}(z,r)}(\overline{x})$ denote the output of an ideal execution (by Definition α) with a trusted party computing f, the input series $\overline{x}$ and where the ideal model adversary $\tilde{\mathcal{S}}$'s auxiliary input and random tape equal z and r, respectively. Furthermore, for any subset $J\subseteq[n]$, let $\text{IDEAL}^{(\alpha)}_{f,I,\tilde{\mathcal{S}}(z,r)}(\overline{x})|_J$ be the restriction of $\text{IDEAL}^{(\alpha)}_{f,I,\tilde{\mathcal{S}}(z,r)}(\overline{x})$ to the output of $\tilde{\mathcal{S}}$ and the parties in J. Finally, let $J=J_{\tilde{\mathcal{S}}(z,r)}(\overline{x})$ be the set of parties in the ideal execution with $\tilde{\mathcal{S}}$ who do not output $\perp$ (this set is a deterministic function of z, r and $\overline{x}$), and let $J_{\tilde{\mathcal{S}}(z)}(\overline{x})$ denote the associated random variable when r is chosen uniformly. Now, we claim that for every set I, every auxiliary input z and random tape r (for S or $\tilde{\mathcal{S}}$), and every set of inputs $\overline{x}$,

$$\text{IDEAL}^{(5)}_{f,I,\tilde{\mathcal{S}}(z,r)}(\overline{x})|_{J_{\tilde{\mathcal{S}}(z,r)}(\overline{x})}=\text{IDEAL}^{(2)}_{f,I,\mathcal{S}(z,r)}(\overline{x})|_{J_{\tilde{\mathcal{S}}(z,r)}(\overline{x})} \tag{2}$$

where $\mathcal{S}$ and $\tilde{\mathcal{S}}$ are invoked with auxiliary input z and random-tape r. In order to see that Eq. (2) holds, notice the following. First, $\tilde{\mathcal{S}}$ (upon auxiliary input z, random tape r and inputs $\{x_i\}_{i\in I}$)

[9] This is the point in the proof where we need the original protocol to be such that all parties terminate simultaneously. Actually, it suffices that each party defines its output at a fixed and predetermined point. However, in such a case, simultaneous termination can anyway be achieved.

sends exactly the same inputs to the trusted party as $\mathcal{S}$ does (upon auxiliary input z, random tape r and inputs $\{x_i\}_{i\in I}$). Next, 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. In addition, $\tilde{\mathcal{S}}$ outputs exactly the same string that $\mathcal{S}$ outputs. Eq. (2) therefore follows.

By assumption, Π is computationally t-secure with unanimous abort and partial fairness. It therefore holds that for every set $I \subset [n]$ such that $|I| < t$, and for *every set* $J \subseteq [n]$

$$\left\{\textsc{ideal}^{(2)}_{f,I,\mathcal{S}(z)}(\overline{x})|_J\right\}_{\overline{x},z} \stackrel{c}{\equiv} \left\{\textsc{real}_{\Pi,I,\mathcal{A}(z)}(\overline{x})|_J\right\}_{\overline{x},z}$$

Next, notice that the sets $J_{\tilde{\mathcal{S}}}$ and $J_{\tilde{\mathcal{A}}}$ are fully defined given $\tilde{\mathcal{A}}$ and $\tilde{\mathcal{S}}$'s outputs respectively. (Recall that $J_{\tilde{\mathcal{S}}}$ equals the set of parties not outputting $\perp$ in an ideal execution with $\tilde{\mathcal{S}}$, and $J_{\tilde{\mathcal{A}}}$ equals the set of parties not outputting $\perp$ in a real execution of $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$.) Furthermore, by the definitions of $\mathcal{A}$ and $\mathcal{S}$, it follows that their outputs also fully define $J_{\tilde{\mathcal{S}}}$ and $J_{\tilde{\mathcal{A}}}$. Therefore, $J_{\tilde{\mathcal{S}}}$ (resp., $J_{\tilde{\mathcal{A}}}$) is part of the global output $\textsc{ideal}$ (resp., $\textsc{real}$). This implies that,

$$\left\{\textsc{ideal}^{(2)}_{f,I,\mathcal{S}(z)}(\overline{x})|_{J_{\tilde{\mathcal{S}}(z)}(\overline{x})}\right\} \stackrel{c}{\equiv} \left\{\textsc{real}_{\Pi,I,\mathcal{A}(z)}(\overline{x})|_{J_{\tilde{\mathcal{A}}(z)}(\overline{x})}\right\} \tag{3}$$

(Otherwise, we could distinguish $\textsc{ideal}^{(2)}_{f,I,\mathcal{S}(z)}(\overline{x})$ from $\textsc{real}_{\Pi,I,\mathcal{A}(z)}(\overline{x})$ by considering the restriction to $J_{\tilde{\mathcal{S}}}$ or to $J_{\tilde{\mathcal{A}}}$, respectively.) Combining Eq. (3) with Equations (1) and (2), we have that

$$\left\{\textsc{ideal}^{(5)}_{f,I,\tilde{\mathcal{S}}(z)}(\overline{x})|_{J_{\tilde{\mathcal{S}}(z)}(\overline{x})}\right\} \stackrel{c}{\equiv} \left\{\textsc{real}_{\tilde{\Pi},I,\tilde{\mathcal{A}}(z)}(\overline{x})|_{J_{\tilde{\mathcal{A}}(z)}(\overline{x})}\right\}$$

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,

$$\left\{\textsc{ideal}^{(5)}_{f,I,\tilde{\mathcal{S}}(z)}(\overline{x})\right\} \stackrel{c}{\equiv} \left\{\textsc{real}_{\tilde{\Pi},I,\tilde{\mathcal{A}}(z)}(\overline{x})\right\}$$

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: the only difference between Π' and $\tilde{\Pi}$ is that in Π' the adversary can cause a party P_i to receive $\perp$ in a broadcast with abort execution. Now, when this happens, P_i sends $\perp$ to all honest parties in the next blank round of Π'. However, the adversary for $\tilde{\Pi}$ can just send $\perp$ itself to all the honest parties in the next blank round. The same effect is therefore achieved. Formally:

Lemma 4.2 *Let Π be any protocol in the broadcast model 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 $I \subset [n]$ with $|I| < t$,*

$$\left\{\textsc{real}_{\tilde{\Pi},I,\tilde{\mathcal{A}}(z)}(\overline{x})\right\} \equiv \left\{\textsc{real}_{\Pi',I,\mathcal{A}'(z)}(\overline{x})\right\}$$

Proof: We begin by describing the adversary $\tilde{\mathcal{A}}$. Intuitively, $\tilde{\mathcal{A}}$ works by simulating the executions of *broadcast with abort* for $\mathcal{A}'$. If a party receives $\perp$ in Π' (in a broadcast with abort execution or in a blank round), then $\tilde{\mathcal{A}}$ sends the appropriate $\perp$ messages in a blank round of $\tilde{\Pi}$. This strategy works because in Π', a party's output is the same if it receives $\perp$ in a broadcast with abort execution or in the blank round that immediately follows it. Formally, adversary $\tilde{\mathcal{A}}$ invokes $\mathcal{A}'$ and in every round of the execution of $\tilde{\Pi}$ works as follows:

1. *Receiving messages in rounds* $r, r+1$*:* $\tilde{\mathcal{A}}$ receives the broadcast and point-to-point messages from the honest parties in $\tilde{\Pi}$. For every message broadcast by an honest party, $\tilde{\mathcal{A}}$ simulates a "broadcast with abort" execution, playing the honest parties' roles (where $\mathcal{A}'$ plays the corrupted parties' roles). In addition, $\tilde{\mathcal{A}}$ forwards any point-to-point messages unchanged to $\mathcal{A}'$.

2. *Sending messages in round* $r, r+1$*:* $\tilde{\mathcal{A}}$ plays the honest parties' roles in "broadcast with abort" executions, in which $\mathcal{A}'$ broadcasts 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 in $\tilde{\Pi}$. However, if at least one honest party outputs a message m, then $\tilde{\mathcal{A}}$ broadcasts m. As before, point-to-point messages are forwarded unchanged.

3. *Blank round following round* $r+1$*:* If an honest party P_i receives $\perp$ in the simulated "broadcast with abort" execution of rounds r and $r+1$, then $\tilde{\mathcal{A}}$ sends $\perp$ to *all* the honest parties 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 broadcast with abort. This means that some parties may receive $\perp$ instead of the broadcasted message. However, in this case, $\tilde{\mathcal{A}}$ knows that this event occurred and can send $\perp$ to all the honest parties in the following blank round, as would occur in an execution of Π'. The key point is that in Π' it *makes no difference* to an honest parties if it received $\perp$ in a broadcast with abort execution or in the following blank round. We conclude that 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 Π'. ■

Concluding the proof of Theorem 1: 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 every real-model adversary $\tilde{\mathcal{A}}$ for $\tilde{\Pi}$, there exists an ideal-model simulator $\tilde{\mathcal{S}}$ such that the output distribution of a real execution with $\tilde{\mathcal{A}}$ is computationally indistinguishable (or statistically close) to an ideal execution with abort and no fairness with $\tilde{\mathcal{S}}$. 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 abort and no fairness with $\tilde{\mathcal{S}}$. That is, Π' is t-secure with abort and no fairness, as required. ■

The complexity of protocol Π': We remark that the transformation of Π to Π' preserves the round complexity of Π. In particular, the number of rounds in Π' equals exactly 3 times the number of rounds in Π. Regarding the communication complexity (i.e., bandwidth), this is the same in Π and Π' except for the cost incurred in simulating the broadcast channel. Notice that in the "broadcast with abort" protocol, if a dealer sends a k-bit message, then the total bandwidth equals $n \cdot k$. (If the dealer cheats and sends "long" messages, then the bandwidth is upper-bound by the length of the longest acceptable message times n.) Therefore, the number of bits sent in an execution of Π' is at most n times that sent in an execution of Π.

5 Secure Computation with Abort and Partial Fairness

In this section we show that for any functionality f, there exists a protocol for the *computational* t-secure computation of f with abort and partial fairness, for any t (assuming the existence of trapdoor permutations). Furthermore, for any functionality f, there exists a protocol for the *information-theoretic* $n/2$-secure computation of f with abort and partial fairness (and without any complexity assumptions). All protocols are in the point-to-point communication model, and assume no broadcast channel or primitive.

Outline: We begin by motivating why the strategy used in Section 4 to obtain secure computation with abort does not provide partial fairness. The problem lies in the fact that due to the use of a "broadcast with abort" protocol (and not a real broadcast channel), the adversary can disrupt messages that are broadcast by honest parties. Now, in the definition of security with abort and partial fairness, once an honest P_1 receives its output, all honest parties must receive their output. Therefore, the adversary must not be allowed to disrupt the communication following the time that an honest P_1 receives its output. If broadcast is utilized in the remainder of the protocol (as is indeed the case for known protocols), then the adversary will be able to disrupt the communication and prevent all other honest parties from receiving output.

We solve this problem here by having the parties compute a different functionality. This functionality is such that once P_1 gets its output, it can supply all the other parties with their output directly and without broadcast. Of course, this functionality must not reveal anything about the other parties' outputs to P_1. As a first attempt, consider what happens if instead of computing the original functionality f, the parties first compute the following:

First attempt:

Inputs: $\overline{x} = (x_1, \ldots, x_n)$

Outputs:

- Party P_1 receives its own output $f_1(\overline{x})$. In addition, for every $i > 1$, it receives $c_i = f_i(\overline{x}) \oplus r_i$ for a uniformly distributed r_i.
- For every $i > 1$, party P_i receives the string r_i.

That is, for each $i > 1$, party P_i receives a random pad r_i and P_1 receives an "encryption" of $f_i(\overline{x})$ with that random pad. Now, assume that the parties use a protocol that is secure with abort and *no* fairness in order to compute this new functionality. Then, there are two possible outcomes to such a protocol execution: either all parties receive their prescribed output, or at least one honest party receives $\perp$. In the case that at least one honest party receives $\perp$, this party can notify P_1 who can then immediately halt. The result is that no parties, including the corrupted ones, receive output (if P_1 does not send the c_i values, then the parties only obtain r_i which contains no information on $f_i(\overline{x})$). In contrast, if all parties received their prescribed output, then party P_1 can send each party P_i its encryption c_i, allowing it to reconstruct the output by computing $f_i(\overline{x}) = c_i \oplus r_i$. The key point is that the adversary is unable to prevent P_1 from sending these c_i values and no broadcast is needed in this last step. Of course, if P_1 is corrupted, then it will learn all the corrupted parties' outputs first. However, under the definition of partial fairness, this is allowed.

The flaw in the above strategy arises in the case that P_1 is corrupted. Specifically, a corrupted P_1 can send the honest parties modified values, causing them to conclude with incorrect outputs. This is in contradiction to what is required of secure computation. Therefore, we modify the functionality that is computed so that a corrupted P_1 is unable to cheat. In particular, the aim is to prevent the adversary from modifying $c_i = f_i(\overline{x}) \oplus r_i$ without P_i detecting this modification. If the adversary can be restrained in this way, then it can choose not to deliver an output; however, any output delivered is guaranteed to be correct. The above-described aim can be achieved using standard (information-theoretic) authentication techniques, based on pairwise independent hash functions. That is, let $\mathcal{H}$ be a family of pairwise independent hash functions $h : \{0,1\}^k \rightarrow \{0,1\}^k$. Then, the functionality that the parties compute is as follows:

Functionality F**:**

Inputs: $\overline{x} = (x_1, \ldots, x_n)$

Outputs:

- Party P_1 receives its own output $f_1(\overline{x})$. In addition, for every $i > 1$, it receives $c_i = f_i(\overline{x}) \oplus r_i$ for a uniformly distributed r_i, and $a_i = h_i(c_i)$ for $h_i \in_R \mathcal{H}$. (Note, P_1 receives a_i but not the function description h_i.)
- For every $i > 1$, party P_i receives the string r_i and the description of the hash function h_i.

Notice that as in the first attempt, P_1 learns nothing of the output of any honest P_i (since $f_i(\overline{x})$ is encrypted with a random pad). Furthermore, if P_1 attempts to modify c_i to c_i' in any way, then the probability that it will generate the correct authentication value $a_i' = h_i(c_i')$ is at most 2^{-k} (by the pairwise independent properties of h_i). Thus, the only thing a corrupt P_1 can do is refuse to deliver the output. We now formally prove the above:

Theorem 4 *For any probabilistic polynomial-time n-ary functionality f, there exists a protocol in the point-to-point model for the computational t-secure computation of f* with abort and partial fairness, *for any t. Furthermore, there exists a protocol in the point-to-point model for the information-theoretic $n/2$-secure computation of f* with abort and partial fairness.

Proof: We begin by describing the protocol for computing f, as motivated above.

Protocol 4 (protocol for secure computation with abort and partial fairness for any f):

1. Stage 1 – computation: *The parties use any protocol for secure (computational or information-theoretic) computation with abort and* no fairness *in order to compute the functionality F defined above.*[10] *Thus, P_1 receives $f_1(\overline{x})$ and a pair (c_i, a_i) for every $i > 1$, and each P_i $(i > 1)$ receives (r_i, h_i) such that $c_i = f_i(\overline{x}) \oplus r_i$ and $a_i = h_i(c_i)$.*
2. Stage 2 – blank round: *After the above protocol concludes, a blank-round is added so that if any party receives $\perp$ for its output from Stage 1, then it sends $\perp$ to P_1 in this blank round.*
3. Stage 3 – outputs: *If P_1 received any $\perp$-messages in the blank round, then it sends $\perp$ to all parties and halts outputting $\perp$. Otherwise, for every i, it sends (c_i, a_i) to P_i and halts, outputting $f_1(\overline{x})$.*

   Party P_i outputs $\perp$ if it received $\perp$ from P_1 (it ignores any $\perp$ it may receive from other parties). If it received (c_i, a_i) from P_1 (and not $\perp$), then it checks that $a_i = h_i(c_i)$. If yes, it outputs $f_i(\overline{x}) = c_i \oplus r_i$. Otherwise, it outputs $\perp$.

[10] By Corollaries 2 and 3 in Section 4, appropriate protocols exist. That is, information-theoretic secure protocols exist for $t < n/2$. Furthermore, assuming the existence of trapdoor permutations, computational secure protocols exist for any t.

Intuitively, the security of Protocol 4 with abort and partial fairness is derived from the fact that Stage 1 is run using a protocol that is secure with abort (even though it has no fairness property). Consider the two cases regarding whether or not P_1 is corrupted:

1. P_1 *is corrupt:* in this case, $\mathcal{A}$ receives all the outputs of the corrupted parties first. Furthermore, $\mathcal{A}$ can decide exactly which parties to give output to and which not. However, this is allowed in the setting of secure computation with abort and partial fairness, and so is fine. We stress that $\mathcal{A}$ cannot cause an honest party to output any value apart from $\perp$ or its correct output. This is because the authentication properties of pairwise independent hash functions guarantee that $\mathcal{A}$ does not modify c_i, and the correctness of the protocol of Stage 1 guarantees that $c_i \oplus r_i$ equals the correct output $f_i(\overline{x})$.

2. P_1 *is honest:* there are two possible cases here; either some honest party received $\perp$ in the computation of Stage 1 or all honest parties received their correct outputs. If some honest party received $\perp$, then this party sends $\perp$ to P_1 in Stage 2 and thus no parties (including the corrupted parties) receive output. (Similarly, if $\mathcal{A}$ sends $\perp$ to P_1 in Stage 2 then no parties receive output.) In contrast, if all honest parties received their outputs and $\mathcal{A}$ does not send $\perp$ to P_1 in Stage 2, then all parties receive outputs and the adversary cannot cause any honest party to abort. We therefore have that in this case complete fairness is achieved, as required.

In order to formally prove the security of the protocol, we use the sequential composition theorem of [9]. This theorem states that we can consider a model which is a hybrid of the ideal and real models. In this hybrid model, the parties all interact with a trusted party for the computation of Stage 1 (where the ideal model for this computation is that of secure computation with abort and no fairness). Then, Stages 2 and 3 take place as in a real execution. The result is a protocol that is a hybrid of real and ideal executions. In order to prove the security of the (real) protocol, it suffices to construct an ideal-model simulator for the hybrid protocol. Thus, the description of the parties and adversary below relate to this hybrid model (the parties send messages to each other, as in a real execution, and to a trusted party, as in an ideal execution). The proof of [9] is stated for secure computation without abort (and complete fairness); however it holds also for secure computation with abort and no fairness.

Let $\mathcal{A}$ be an adversary attacking Protocol 4 in the above-described hybrid model. We construct an adversary $\mathcal{S}$ who works in the ideal model for f with abort and partial fairness. $\mathcal{S}$ internally incorporates $\mathcal{A}$ and simulates the hybrid execution for $\mathcal{A}$. Therefore, $\mathcal{S}$ has *external* communication with the trusted party of its ideal model and *internal*, simulated communication with the adversary $\mathcal{A}$. In our description of $\mathcal{S}$, we differentiate between the cases that P_1 is corrupt and P_1 is honest:

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, $\mathcal{S}$ externally sends these inputs unmodified to the trusted party of its ideal model for computing f. If the inputs are not valid, then in the hybrid model all parties receive $\perp$ as output. Therefore, $\mathcal{S}$ internally hands $\perp$ to $\mathcal{A}$ as its output from Stage 1 and simulates all honest parties sending $\perp$ in Stage 2 (as would occur in a hybrid execution). $\mathcal{S}$ then halts, outputting whatever $\mathcal{A}$ does. Otherwise, if the inputs are valid, $\mathcal{S}$ receives all the corrupted parties outputs $\{f_i(\overline{x})\}_{i\in I}$ (this is the case because $\mathcal{S}$ controls P_1 and by partial fairness, when P_1 is corrupt the adversary receives the corrupted parties' outputs first). $\mathcal{S}$ then constructs the corrupted parties' outputs from Stage 1 that $\mathcal{A}$ expects to see in the hybrid execution. $\mathcal{S}$ defines P_1's output as follows: First, P_1's personal output is $f_1(\overline{x})$. Next, for every corrupted party P_i, party P_1's output contains the pair $(c_i = f_i(\overline{x}) \oplus r_i, h_i(c_i))$ for $r_i \in_R \{0,1\}^k$ and $h_i \in_R \mathcal{H}$. Finally, for every honest party P_j, party P_1's output contains

a pair (c_j, a_j) where $c_j, a_j \in_R \{0,1\}^k$. This defines P_1's output. We now define how $\mathcal{S}$ constructs the other corrupted parties' outputs: for every corrupted P_i, simulator $\mathcal{S}$ defines P_i's output to equal (r_i, h_i) where these are the values used in preparing the corresponding pair $(c_i, h_i(c_i))$ in P_1's output. (We note that $\mathcal{S}$ can prepare these values because it knows $f_i(\overline{x})$ for every corrupted party P_i.) $\mathcal{S}$ then internally passes $\mathcal{A}$ all of these outputs. In the hybrid model, after receiving the outputs from Stage 1, $\mathcal{A}$ sends a set J' to the trusted party instructing it to give outputs to the parties specified in this set (all other parties receive $\perp$). $\mathcal{S}$ obtains this set J' from $\mathcal{A}$ and records it.

$\mathcal{S}$ continues by simulating P_l sending $\perp$ to P_1 in the blank round, for every honest party P_l for which $l \notin J'$ (as would occur in a hybrid execution). Then, in the last stage, $\mathcal{A}$ (controlling P_1) sends to each honest party P_j a pair (c'_j, a'_j) or $\perp$. $\mathcal{S}$ 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 (c_j, a_j) that $\mathcal{S}$ gave $\mathcal{A}$ in Stage 1. (These are the parties who do not see $\perp$ in the execution and whose checks of Stage 3 succeed; they therefore do not abort.) $\mathcal{S}$ concludes by externally 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 its trusted party; rather it just records them.[11] (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 before it sends the trusted party the set J' of honest parties receive output from Stage 1. However, $\mathcal{S}$ does *not* have the corrupted parties' outputs yet. Fortunately, when P_1 is not corrupted, $\mathcal{S}$ can perfectly simulate the corrupted parties outputs from Stage 1 by merely providing them with (r_i, h_i) where $r_i \in_R \{0,1\}^k$ and $h_i \in_R \mathcal{H}$. After internally passing $\mathcal{A}$ the simulated corrupted parties' outputs, $\mathcal{S}$ obtains a set J' from $\mathcal{A}$, instructing the trusted party of the hybrid model which parties should receive output.

   $\mathcal{S}$ continues by simulating Stages 2 and 3 of the protocol. As above, $\mathcal{S}$ simulates every honest party P_l for which $l \notin J'$ sending $\perp$ to P_1 in Stage 2. Furthermore, $\mathcal{S}$ obtains any messages sent by $\mathcal{A}$ in this stage. If $\mathcal{A}$ sends $\perp$ to P_1 in Stage 2, then $\mathcal{S}$ simulates P_1 sending $\perp$ to all parties, sends invalid inputs to the trusted party and halts. Likewise, if J' does not contain *all* the honest parties, then $\mathcal{S}$ internally simulates P_1 sending $\perp$ to all the corrupted parties, and externally sends invalid inputs to the trusted party. (These cases correspond to the case that no parties receive their prescribed output.)

   In contrast, if J' contains all the honest parties (i.e., no honest party received $\perp$ from Stage 1) and $\mathcal{A}$ did not send $\perp$ to P_1 in Stage 2 of the simulation, then $\mathcal{S}$ externally sends the trusted party the inputs that it recorded from $\mathcal{A}$ above, receiving back all of the corrupted parties outputs $\{f_i(\overline{x})\}_{i \in I}$. Then, for each corrupted party's output $f_i(\overline{x})$, simulator $\mathcal{S}$ generates the pair that corrupted P_i would see in a hybrid execution. In particular, previously in the simulation $\mathcal{S}$ provided P_i with a pair (r_i, h_i) where $r_i \in_R \{0,1\}^k$ and $h_i \in_R \mathcal{H}$. Now, $\mathcal{S}$ simulates P_1 sending corrupted party P_i the pair (c_i, a_i) where $c_i = f_i(\overline{x}) \oplus r_i$ and $a_i = h_i(c_i)$. ($\mathcal{S}$ can do this because it knows the random-pad r_i and the hash function h_i.) Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ does and halts.

[11] $\mathcal{S}$ cannot forward the inputs to the trusted party yet, because in the model of partial fairness as soon as it does this all parties receive output. However, in the execution of Protocol 4, $\mathcal{A}$ can cause the execution to abort at a later stage.

The fact that the global output in the ideal execution with $\mathcal{S}$ is identically distributed to the global output in a hybrid execution with $\mathcal{A}$ is derived from the following observations. First, $\mathcal{A}$'s outputs from Stage 1 can be perfectly simulated, both when P_1 is corrupt and when P_1 is honest. Second, 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, with overwhelming probability, 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 pair (c_j, a_j) that it received as output from Stage 1 (and who did not see $\perp$ at any time in the execution). This is due to the authentication properties of pairwise independent hash functions. Likewise, in the ideal-model simulation, $\mathcal{S}$ designates these same parties to be the ones receiving output. Therefore, except with negligible probability, 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 abort in the hybrid model corresponds to this case in the real model (and likewise for the case that all parties receive output). This completes the proof of Theorem 4 ■

6 Obtaining Security Under Composition

As we have discussed, one of our main aims in removing the reliance on a broadcast channel in secure protocols was to remove the obstacle that such a channel poses when security under composition is considered. (Of course, achieving secure computation without a trusted setup phase, as needed for authenticated Byzantine Agreement, is also of importance. However, in a setting where *only point-to-point communication* is available, the possibility of obtaining secure protocols for $t \geq n/3$ that remain secure under parallel or concurrent composition was unknown, even assuming a trusted setup phase.) In this section we *informally discuss* the ramifications of our results on protocol composition. We consider both parallel composition (for the case of no honest majority) and concurrent composition (when there is an honest majority). The type of composition that we consider here is self composition, where the same protocol is run by the same set of parties many times. Furthermore, each party plays the same role in each execution (as in zero-knowledge where the same party plays the prover in all executions and likewise for the verifier).

As we have shown, the protocol compiler of Construction 3 is such that the only difference between the original protocol that uses a broadcast channel and the resulting protocol that uses only point-to-point channels is with respect to the unanimity of abort. The formal proof of this fact was demonstrated in the stand-alone model only. However, it is not hard to see that all the claims also go through in the setting of composition. Therefore, it suffices to demonstrate the existence of protocols in the broadcast model that remain secure under composition in order to derive the existence of analogous protocols in the point-to-point network model. The presentation here is very informal; in particular, we do not even formally define what is meant by security under composition. We now present the results:

Concurrent self composition for $t < n/2$. The broadcast-model protocol of [37] for the information-theoretic $n/2$-secure computation of any functionality has been shown to compose concurrently [10].[12] Therefore, by applying the protocol compiler of Construction 3 to the protocol of [37], we obtain a protocol that provides information-theoretic $n/2$-secure computation (with abort and no fairness) in the point-to-point model, and remains secure under concurrent composition. Next, using the transformation in the proof of Theorem 4, we obtain an analogous protocol with abort and partial fairness.[13]

Parallel self composition for $t \geq n/2$. A variant of the protocol of [26] for the broadcast model can be shown to compose in parallel. Specifically, consider a variant of [26] where the only rewinding that is carried out by the ideal-model simulator is in simulating zero-knowledge proofs and extracting witnesses from proofs of knowledge (we note that the protocol as described in [23] has this property). Then, if the zero-knowledge proofs and proofs of knowledge that are used are such that they remain secure under parallel composition, it follows that the entire secure protocol composes in parallel. It remains to demonstrate the existence of zero-knowledge proofs and proofs of knowledge that compose in parallel. The fact that there exist zero-knowledge proofs that compose in parallel was proven in [24]. In contrast, we do not know of any published theorem stating the existence of a zero-knowledge proof of knowledge that composes in parallel (i.e., for which knowledge extraction succeeds even in the setting of parallel composition). Nevertheless, it is not difficult to show that the knowledge extractors for standard zero-knowledge proofs of knowledge of $\mathcal{NP}$ (e.g., [7, 25]) succeed under parallel composition. Given this observation, it is possible to obtain zero-knowledge proofs of knowledge for which both the simulation and extraction succeed under parallel composition. This can be achieved using ideas from the protocol of [18] (where the only rewinding carried out by the simulator is in running a knowledge extractor). We thus conclude that in the broadcast model, there exist secure protocols for $t \geq n/2$ that compose in parallel. By applying Construction 3 and the transformation in the proof of Theorem 4 to such protocols, we obtain secure protocols with abort and partial fairness in the point-to-point model that *compose in parallel*. We remark that general secure computation under concurrent composition cannot be achieved for the case of $t \geq n/2$, irrespective of the use of a broadcast channel [32].

Universal composability for any t. In Section 3.2, we showed that the ideal broadcast functionality can be securely realized in the framework of universal composability. Recall that in this framework, a completely asynchronous point-to-point network is considered and output delivery is not guaranteed. We now apply Proposition 3.2 (proven in Section 3.2) in order to obtain universally composable secure computation without a broadcast channel.

We first consider the scenario where a majority of the parties are honest, but this majority may be less than 2/3 (i.e., $n/3 \leq t < n/2$). For this range of corruptions, [10] (building on [37]) showed that universally composable protocols exist for any functionality, assuming that the parties are given access to an ideal broadcast functionality. Combining this with Proposition 3.2 we have that universally composable protocols exist for any functionality in the point-to-point model.

Next, we consider the setting of an honest *minority*. In this setting, universal composability cannot be achieved in the plain model [12, 10]. Nevertheless, it has been shown that in the common reference string model,[14] it is possible to securely compute any functionality in an asynchronous

[12] Actually, this claim does not yet have an accompanying proof.

[13] We remark that Theorem 4 actually provides a transformation of *any* generic secure protocol. That is, any generic protocol for the broadcast model that is secure with abort and no fairness can be transformed into a general protocol for the point-to-point model, where the resulting protocol is secure with abort and partially fair. The theorem was not stated in this way so that it would not be overly cumbersome.

[14] In this model, all the parties are given a common, public reference string that is ideally chosen from a given

network with a *broadcast channel* [14]. Therefore, by combining this with Proposition 3.2, we obtain universally composable protocols for any multi-party functionality in an asynchronous point-to-point network augmented by a common reference string. We remark that the lower bound of [33] with respect to the composition of authenticated Byzantine agreement holds also in the common reference string model.

We conclude by noting that the above results on the composition of secure protocols are incomparable. On the one hand, the notion of composition considered in the framework of universal composability is very strong. In particular, it guarantees security when a protocol is run concurrently with arbitrary other protocols and arbitrary sets of parties. However, this result requires a common reference string to achieve. On the other hand, our protocols for (concurrent and parallel) self composition achieve a far weaker notion of security; in self composition, a single protocol is run many times by a single set of parties, and this is the only protocol being run in the network. Furthermore, for the case of $t \geq n/2$, only parallel composition is obtained. Nevertheless, the advantage of these protocols over the protocol for universal composability is that they do not require a common reference string and can be run in the plain model.

Acknowledgements

We would like to thank Oded Goldreich for many helpful discussions, and Juan Garay for pointing out the appropriate reference to [5].

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 Faulty Majority. In *30th FOCS*, pages 468–473, 1989.

[3] D. Beaver and S. Haber. Cryptographic Protocols Provably Secure Against Dynamic Adversaries. In *EUROCRYPT'92,* Springer-Verlag (LNCS 658), pages 307–323, 1992.

[4] D. Beaver, S. Micali and P. Rogaway. The Round Complexity of Secure Protocols. In *22nd STOC*, pages 503–513, 1990.

[5] M. Ben-Or and R. El-Yaniv. Optimally-Resilient Interactive Consistency in Constant Time. *Distributed Computing,* 16(4):249–262, 2003.

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

[7] M. Blum. How to Prove a Theorem So No One Else Can Claim It. *Proceedings of the International Congress of Mathematicians*, Berkeley, California, USA, 1986, pp. 1444-1451.

[8] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *3rd PODC*, pages 154–162, 1984.

distribution.

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

[10] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd FOCS*, pages 136–145, 2001.

[11] R. Canetti, U. Feige, O. Goldreich and M. Naor. Adaptively Secure Multi-Party Computation. In 28*th STOC*, pages 639–648, 1996.

[12] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO 2001*, Springer-Verlag (LNCS 2139), pages 19–40, 2001.

[13] R. Canetti and H. Krawczyk. Universally Composable Notions of Key-Exchange and Secure Channels. In *EUROCRYPT 2002*, Springer-Verlag (LNCS 2332), pages 337–351, 2002.

[14] R. Canetti, Y. Lindell, R. Ostrovsky and A. Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In 34*th STOC*, pages 494–503, 2002.

[15] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *25th STOC*, pages 42–51, 1993.

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

[17] D. Dolev. The Byzantine Generals Strike Again. *Journal of Algorithms,* 3(1):14–30, 1982.

[18] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. In *CRYPTO'89*, Springer-Verlag (LNCS 435), pages 526–544, 1989.

[19] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine agreement. *SIAM Journal of Computing*, 26(2):873–933, 1997.

[20] M. Fitzi, N. Gisin, U. Maurer and O. Von Rotz. Unconditional Byzantine Agreement and Multi-Party Computation Secure Against Dishonest Minorities from Scratch. In *EUROCRYPT 2002,* Springer-Verlag (LNCS 2332), pages 482–501, 2002.

[21] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Byzantine Agreement Secure Against Faulty Majorities From Scratch. In 21*st PODC,* pages 118–126, 2002.

[22] Z. Galil, S. Haber and M. Yung. Cryptographic Computation: Secure Fault Tolerant Protocols and the Public Key Model. In *CRYPTO'87,* Springer-Verlag (LNCS 293), pages 135–155, 1987.

[23] O. Goldreich. *Secure Multi-Party Computation*. Manuscript, version 1.4, 2002. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[24] O. Goldreich. Concurrent Zero-Knowledge With Timing Revisited. In 34*th STOC*, pages 332–340, 2002.

[25] O. Goldreich, S. Micali and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, 38(1):691–729, 1991.

[26] 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 [23].

[27] 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.

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

[29] J. Kilian, A General Completeness Theorem for Two-Party Games. In *23rd STOC*, pages 553- 560, 1991.

[30] L. Lamport. The Weak Byzantine Generals Problem. *Journal of the ACM*, 30(3):668–676, 1983.

[31] L. Lamport, R. Shostack, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[32] Y. Lindell. Lower Bounds for Concurrent Self Composition. To appear in the *1st Theory of Cryptography Conference* (TCC), 2004.

[33] Y. Lindell, A. Lysyanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In 34*th STOC*, pages 514–523, 2002.

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

[35] M. Pease, R. Shostak and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

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

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

[38] 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 universal composability; for more details see [10]. The framework provides a rigorous 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., [27, 34, 1, 9]), 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 any adversary can gain nothing more from 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. In other words, we require that a real execution can be *emulated* in the above ideal process (where the meaning of *emulation* is described below). 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, plus an adversary $\mathcal{A}$ that controls the communication channels and potentially corrupts parties. Emulation means 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. 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 [10] 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 [10]: 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 activation 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.

The standard network model considered for this framework is one where the adversary sees all the messages sent, and delivers or blocks these messages at will. Note that although the adversary may block messages, it cannot modify messages sent by honest parties (i.e., the communication lines are ideally authenticated). Thus, real executions take place in an ideally authenticated asynchronous network. However, unlike other ideal model definitions, the adversary also delivers (or blocks) messages between the honest parties and the trusted party/ideal functionality. Among other things, this implies that output delivery is not guaranteed and that there is no fairness.