Secure Computation Without Agreement

Shafi Goldwasser[*] Yehuda Lindell[†]

July 14, 2002

Abstract

It has recently been shown that executions of authenticated Byzantine Agreement protocols in which more than a third of the parties are corrupted, cannot be composed concurrently, in parallel, or even sequentially (where the latter is true for deterministic protocols). This result puts into question any usage of authenticated Byzantine agreement in a setting where many executions take place. In particular, this is true for the whole body of work of secure multi-party protocols in the case that 1/3 or more of the parties are corrupted. Such protocols strongly rely on the extensive use of a broadcast channel, which is in turn realized using authenticated Byzantine Agreement. Essentially, this use of Byzantine Agreement cannot be eliminated since the standard definition of secure computation (for the case that less than 1/2 of the parties are corrupted) actually implies Byzantine Agreement. Moreover, it is accepted folklore that the use of a broadcast channel is essential for achieving secure multiparty computation, when 1/3 or more of the parties are corrupted.

In this paper we show that this folklore is false. We mildly relax the definition of secure computation allowing abort, and show how this definition can be reached. The difference between our definition and previous ones is as follows. Previously, if one honest party aborted then it was required that all other honest parties also abort. Thus, the parties *agree* on whether or not the protocol execution terminated successfully or not. In our new definition, it is possible that some parties abort while others receive output. Thus, there is no agreement regarding the success of the protocol execution. We stress that in all other aspects, our definition remains the same. In particular, if an output is received it is guaranteed to have been computed correctly. The novelty of the new definition is in *decoupling* the issue of agreement from the central security issues of privacy and correctness in secure computation. As a result the lower bounds of Byzantine Agreement no longer apply to secure computation. Indeed, we prove that secure multi-party computation 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). An important corollary of our result is the ability to obtain multi-party protocols that compose.

1 Introduction

In the setting of secure multi-party computation, a set of n parties with private inputs wish to jointly and securely perform a joint 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 encompasses computations as simple as coin-tossing and agreement, and as

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

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

complex as electronic voting, electronic auctions, electronic cash schemes, anonymous transactions, and private information retrieval schemes.

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. These definitions aimed to ensure a number of important security properties. The most central of these are:

- *Privacy:* No party should learn anything more that its prescribed output.
- *Correctness:* Each party is guaranteed that the output that it receives is correct.
- *Independence of Inputs:* Corrupted parties' inputs are committed to independently of honest parties' inputs.
- *Guaranteed output delivery:* Corrupted parties should not be able to prevent honest parties from receiving their output. (This is not always possible and is therefore not always required.)
- *Fairness:* Corrupted parties should receive an output only if honest parties do. (As with the previous item, this is not always achievable and is therefore is not always fully required .)

The standard definition today [19, 1, 25, 5] formalizes the above requirements in the following 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 the ideal world by having the parties simply send their inputs to the trusted party. This trusted party 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 this ideal computation. A real protocol that is run by the parties (in a world where no trusted party exists) is said to be secure, if no adversary controlling a coalition of corrupted parties can do more harm in a real execution that in the above ideal computation.

Broadcast: In the construction of protocols, the ability to "broadcast" messages (if needed) was assumed as a primitive, where broadcast takes on the meaning of the Byzantine Generals problem [23]. 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:

1. For $t < n/3$, Byzantine agreement is possible by a deterministic protocol with round complexity $O(t)$ [26], and by a probabilistic protocol with expected round complexity $O(1)$ [12];
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 [26, 23]. (This public-key infrastructure is assumed to be setup in a trusted preprocessing phase.) We note that an information theoretic analogue also exists [27]. 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 $t \geq n/3$).

Fairness: As we have mentioned above, fairness is also considered as an important goal in secure computation. Since the basic notion of fairness is not achievable for all values of t, it takes on different meanings for different values of t. We will single out a few forms of fairness. On the one extreme, we have "complete fairness" that guarantees that if a corrupt party gets its output then all honest parties also get their output. On the other extreme, we have "no fairness" in which the adversary always gets its output and has the power to decide whether or not the honest parties also get output. An intermediate notion that we call "partial fairness" singles out a specified party such that if this specified party is honest then complete fairness is achieved. On the other hand, if the specified party is corrupt, then no fairness is achieved. Thus, fairness is partial.

1.2 Feasibility of secure computation

Wide-reaching results, demonstrating the feasibility of secure computation were also presented in the late 80's. The most central of these are as follows:

1. For $t < n/3$, secure multi-party protocols with complete fairness (and guaranteed output delivery), can be achieved in a point-to-point network and without any setup assumptions. This can be achieved both in the information theoretic setting assuming private channels [4, 10], and in the computational setting (assuming the existence of trapdoor permutations).[1]

2. For $t < n/2$, secure multi-party protocols with complete fairness (and guaranteed output delivery) can be achieved assuming the existence of a broadcast channel. This can be achieved in the infomation theoretic setting [28] and in the computational setting [18] with the same assumptions as above. 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).

3. For $t \geq n/2$, secure multi-party protocols with partial fairness can be achieved assuming a broadcast channel or a trusted pre-processing phase (as in case (2)), and in addition the existence of oblivious transfer [18, 20, 21]. Some works attempting to provide higher levels of fairness (e.g., ensuring that the corrupted parties progress at the same rate towards their output as the honest parties) also appeared [29, 16, 19, 2].

   We note that in this case (of $t \geq n/2$) it is impossible to guarantee output delivery (even given a broadcast channel). Therefore, this property is not required (and some parties may not receive output at all).

We note that all of the above results consider a stand-alone execution of a multi-party protocol only.

1.3 Byzantine agreement and secure computation

There is a close connection between Byzantine agreement and secure multi-party computation. First, Byzantine agreement (or broadcast) is used as a basic and central tool in the construction of secure protocols. In particular, all the feasibility results above assume a broadcast channel (and implement it using Byzantine agreement or authenticated Byzantine agreement). Second, Byzantine agreement is actually a special case of secure computation (this holds by the standard definition

[1]The protocol of [18] uses oblivious transfer which can in turn be constructed from trapdoor permutations. Alternatively, one can transform the protocol of [4] to the computational model by encrypting all messages sent between players with public-key encryption. This transformation assumes the existence of public-key encryption only.

taken for the case that $t < n/2$ where output delivery is guaranteed). Therefore, all the lower bounds relating to Byzantine agreement immediately apply to secure multi-party computation. In particular, the Byzantine agreement problem cannot be solved for any $t \geq n/3$ [26]. Thus, it is also impossible to achieve secure computation with guaranteed output delivery in a point-to-point network for $t \geq n/3$. On the other hand, for $t < n/2$ it *is* possible to obtain secure computation with guaranteed output delivery assuming a broadcast channel. This means that in order to achieve such secure computation for the range of $n/3 \leq t < n/2$, either a physical broadcast channel or a trusted pre-processing phase for running authenticated Byzantine agreement *must* be assumed.

More recently, it was shown that authenticated Byzantine agreement cannot be composed (concurrently or even in parallel), unless $t < n/3$ [24]. This has the following ramifications. On the one hand, in the range of $n/3 \leq t < n/2$, it is *impossible* to obtain secure computation that composes without using a physical broadcast channel. This is because such a protocol in the point-to-point network model and with trusted pre-processing would imply authenticated Byzantine agreement that composes. On the other hand, as we have mentioned, in the range of $t \geq n/2$ the definitions of secure computation do not imply Byzantine agreement. Nevertheless, all protocols for secure computation in this range make extensive use of a broadcast primitive. The impossibility of composing authenticated Byzantine agreement puts this whole body of work into question when composition is required. Specifically without using a physical broadcast channel, none of these protocols compose (even in parallel). In summary, the current state of affairs is that there are no protocols for secure computation in a point-to-point network that compose in parallel or concurrently, for any $t \geq n/3$. Needless to say, the requirement of a physical broadcast channel is very undesirable (and often unrealistic).

1.4 Our Results

We present a mild relaxation of the standard definition of secure multi-party computation that decouples the issue of *agreement* from the issue of *secure multi-party computation.* In particular, our definition focuses on the central issues of privacy and correctness. Loosely speaking, our definition is different in the following way. As we have mentioned, for the case of $t \geq n/2$, it is impossible to guarantee output delivery and therefore some parties may conclude with a special abort symbol $\perp$, and not with their output. Previously [17], it was required that either *all* honest parties receive their outputs or *all* honest parties output $\perp$.[2] Thus the parties all *agree* on whether or not output was received. On the other hand, in our definition some honest parties may receive output while some receive $\perp$, and the requirement of agreement is removed. We stress that this is the only difference between our definition and the previous ones.

We show that it is possible to achieve secure computation according to the new definition for *any* $t < n$ and *without* a broadcast channel or setup assumption (assuming the same computational assumptions made, if any, by corresponding protocols that did use broadcast channels.) Thus, the lower bounds for Byzantine agreement indeed do not imply lower bounds for secure multi-party computation. We note that our result holds in both the information theoretic and the computational models.

A hierarchy of definitions. In order to describe our results in more detail, we present a hierarchy of definitions for secure computation. All the definition fulfill the properties of privacy and

[2]We note that in private communication, Goldreich stated that the requirement in [17] of having all parties abort or all parties receive output was only made in order to simplify the definition.

correctness. The hierarchy that we present here relates to the issues of abort (or failure to receive output) and fairness.

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

   (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). On the other hand, 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 problem that may arise is that of an external party "hacking" into the machine of one 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 (i.e., there is no specified party).

   We stress that in all the above three definitions, if one honest party aborts then so do all honest parties, and thus all 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 one immediately preceding it, 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 Goldreich [17] for the case of any $t < n$ is that of secure computation with unanimous abort and partial fairness. Our new definition is that of secure computation with abort, and as we have mentioned, its key feature is a decoupling of the issues of secure computation and agreement (or unanimity).

Achieving secure computation with abort. Using the terminology introduced above, our results 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. First, we define a weak variant of the Byzantine Generals problem, called *broadcast with abort,* in which not all parties are guaranteed to receive the broadcasted value. In particular, there exists a single value x such that every party either outputs x or aborts. Furthermore, when the broadcasting party is honest, the value x equals its input, similarly to the validity condition of Byzantine Generals. (Notice that in this variant, the parties do not necessarily agree on the output since some may output x while others abort.) We call this "broadcast with abort" because as with secure computation with abort, some parties may output x 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 complete fairness cannot be obtained in this range, even with broadcast). That is, we prove 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* $t < n$ *corruptions.*

The theorem is proved 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 Beaver et al. [3], we obtain the first constant-round protocol in the point-to-point network for the range of $n/3 \leq t < n/2$.[3] 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.*

Composition of secure multi-party protocols. An important corollary of our result is the ability to obtain secure multi-party protocols for $t > n/3$ that compose in parallel or concurrently, without a broadcast channel. 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 [24] authenticated Byzantine Agreement does *not* compose even in parallel. (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.

Recently, Canetti [6] proposed a new framework for multi-party computation that guarantees security in a general setting where many arbitrary protocols are running concurrently. Protocols that fulfill this new definition are called "universally composable". Canetti also proposed studying a communication model where the adversary can block messages sent by the honest parties. In this model, universally composable multi-party computation for any $t < n$ was demonstrated, using a broadcast channel and a common reference string [9]. We show that the broadcast channel used in [9] can be replaced by 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. (We stress that this result does not hold in the standard model where delivery of messages is guaranteed.)

Discussion. We propose that the basic definition of secure computation should focus on the issues of privacy and 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

[3]For the range of $t < n/3$, the broadcast channel in the protocol of [3] can be replaced by the expected constant-round Byzantine agreement protocol of Feldman and Micali [12]. However, there is no known authenticated Byzantine agreement protocol with analogous round complexity.

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 [14, 15] studying a problem similar to ours, although apparently for different motivation.[4] In [14], Fitzi et al. study the question of multi-party computation in the case that the number of faults is $t < n/2$. They show that in this case, it is possible to achieve weak Byzantine agreement (where loosely speaking, either all honest parties abort or all honest parties agree on the broadcasted value). (We note that their protocol is probabilistic and "breaks" the $t < n/3$ lower-bound on deterministic weak Byzantine Agreement protocols of Lamport [22].) 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.

In subsequent work [15], Fitzi et al. studied the question of Byzantine agreement for any $t < n$ and whether its relaxation to weak Byzantine Agreement can be achieved without preprocessing. They show that it is indeed possible to achieve (randomized) weak Byzantine Agreement for *any* $t < n$, in $O(t)$ rounds. 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 with almost no cost. This also results in our obtaining a round-preserving transformation of secure protocols in the broadcast model to those in the point-to-point model. This is in contrast to [14, 15] 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 [15] 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 [15] to $O(r+t)$, while achieving the same level of security.

Recall that Canetti in [6] 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 [6] is different; it aims to decouple the issue of guaranteed output

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

[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 [14, 15]). However, given a protocol that solves a specific secure computation problem, our transformation does not achieve partial fairness. In order to achieve partial fairness, we must revert to a generic protocol. On the other hand, the transformation of [14, 15] works for any protocol. Thus, given a very efficient protocol for a specific problem that achieves partial fairness, it may be "cheaper" to use [14, 15] rather than our results.

delivery from the main issues of secure computation. On the other hand, we focus on the question of agreement by the parties on whether or not output was delivered.

2 Definitions – Secure Computation

In this section we present definitions for secure multi-party computation. The basic description and definitions are based on [17], which in turn follows [19, 1, 25, 5]. 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 to *secure computation without abort.* This is the standard definition used when more than half the parties are honest. According to this definition, all parties receive the output and the adversary cannot disrupt the computation. However, we will not formally present this definition here.

Notation: We denote by U_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; finally, computational indistinguishability is denoted by $\overset{\rm c}{\equiv}$ and statistical closeness by $\overset{\rm s}{\equiv}$. The security parameter is denoted by k.

Multi-party computation. 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 for each party). We refer to such a process as an n-ary functionality and denote it $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$, 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 $i^{\rm th}$ party (with input x_i) is defined to be $f_i(\overline{x})$.

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. Such an adversary may arbitrarily deviate from the specified protocol. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, parties may refuse to participate in the protocol, may substitute their local input (and enter with a different input) and may cease participating in the protocol before it terminates.

Formally, the adversary is modeled by a non-uniform Turing machine: in the computational model this machine is polynomial-time whereas in the information-theoretic model it is unbounded. (We note that by standard arguments, we can assume that the adversary is deterministic.) For simplicity, in this work we consider a *static corruption* model. Therefore, at the beginning of the execution, the adversary is given a set I of corrupted parties which it controls. Then, throughout the computation, the adversary obtains the views of the corrupted parties, and provides them with the messages that they are to send.

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

2.1 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: This definition requires complete fairness. That is, either all parties (including the corrupted parties) receive output or all parties abort. Therefore, the abort is also unanimous. We note that this definition is only achievable when the number of corrupted parties is less than $n/2$ (i.e., $|I| < n/2$), even assuming a broadcast channel. 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 always sends its input x to the trusted party. The corrupted parties may, depending on their inputs $\{x_i\}_{i\in I}$, either abort or send modified 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 honest parties, $x' = x$ always).

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

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

Definition 1 (ideal-model computation with unanimous abort and complete 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 such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$, denoted $\text{IDEAL}^{(1)}_{f,(\mathcal{A},I)}(\overline{x})$, 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 (this definition is usually used when the number of corrupted parties is not limited in any way). Thus, we cannot prevent the "early abort" phenomenon in which the adversary receives its output, whereas the honest parties do not receive theirs. Nevertheless, party P_1 is specified so that if it is honest, then complete fairness is achieved. In contrast, if it 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. 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 always sends its input x to the trusted party. The corrupted parties may, depending on their inputs $\{x_i\}_{i \in I}$, either abort or send modified 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 honest parties, $x' = x$ always).

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

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

In case the first party is corrupted, then for every $i \in I$, the trusted party sends $f_i(\overline{x})$ to party P_i (i.e., the corrupted parties receive their outputs first). Then P_1, depending on the views of all the corrupted parties and controlled by the adversary, 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 $j \notin I$.[6]

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

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 such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$, denoted $\text{IDEAL}^{(2)}_{f,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \ldots, P_n$ and $\mathcal{A}$ resulting from the above described ideal process.*

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: 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 always sends its input x to the trusted party. The corrupted parties may, depending on their inputs $\{x_i\}_{i \in I}$, either abort or send modified 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 honest parties, $x' = x$ always).

[6] An equivalent definition to this one says that P_1 always instructs the trusted party to send the outputs or $\perp$. However, an honest P_1 always instructs the trusted party to provide all parties with outputs. In contrast, a corrupted party can decide whatever it wishes.

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

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 such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$, denoted $\text{IDEAL}^{(3)}_{f,(\mathcal{A},I)}(\overline{x})$, 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 $\perp$. 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 P_1, depending on the views of all the corrupted parties and controlled by the adversary, 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 such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$, denoted $\text{IDEAL}^{(4)}_{f,(\mathcal{A},I)}(\overline{x})$, 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 very similar to the previous one, except that the first party P_1 does not receive its output first. Rather, the adversary always receives the output of the corrupted parties first. Then, it designates which honest parties receive their output and which receive $\perp$. We repeat only the relevant item:

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 $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-ary functionality, where $f = (f_1, \ldots, f_n)$, and let $I \subset [n]$ be such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model *on input vector $\overline{x} = (x_1, \ldots, x_n)$, denoted $\text{IDEAL}^{(5)}_{f,(\mathcal{A},I)}(\overline{x})$, 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). We stress that the messages sent by an honest party in a given round depend on the messages that it received in previous rounds only. On the other hand, the adversary can compute its messages in a given round based on the messages that it receives from the honest parties in the same round. The term rushing refers to this additional adversarial capability.

In this work, we consider a scenario where the parties are connected via a fully connected point-to-point network (and there is no broadcast channel). We refer to this model as the *point-to-point model* (in contrast to the *broadcast model* where the parties 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).[7] In the basic model, we assume that any message sent by an honest party to another honest party is received immediately. However, we also consider a model in which the adversary has control over the delivery of messages. That is, in every round, the adversary can decide to block (i.e., not deliver) some or all of the messages sent between the honest parties.[8] (We stress that since the communication lines are authenticated and private, the only thing that the adversary can do is prevent a message from being sent.) This model of communication is the main model used by Canetti [6] in his work on universally composable security. Finally, we note that we do not assume a preprocessing setup phase.[9]

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

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

[9]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

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

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 such that for every $i \in I$, the adversary $\mathcal{A}$ controls P_i (this is the set of corrupted parties). Then, the* joint execution of Π under $(\mathcal{A}, I)$ in the real model *on input vector $\overline{x} = (x_1, \dots, x_n)$, denoted $\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x})$, is defined as the output vector of $P_1, \dots, P_n$ 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.*

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. On the other hand, in the second, we consider unbounded adversaries and require that the simulation be such that the output distributions of the two models are statistically close.

Definition 7 (computational security): *Let f and Π be as above. We say that protocol Π is a protocol for* computational t-secure computation 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 polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [n]$ with $|I| < t$,*

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

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* information-theoretic t-secure computation with unanimous abort *(resp.,* with abort*) and with* complete fairness *(resp., with* partial fairness *or with* no fairness*), if for every non-uniform*

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 [24], who showed that authenticated Byzantine Agreement does not compose (in parallel or concurrently) when 2/3 or less of the parties are honest. On the other hand, secure channels can be achieved without any limitation on the protocol using them [8]; in particular, without restrictions on composability and the number of faults.

adversary $\mathcal{A}$ for the real model, there exists a non-uniform adversary $\mathcal{S}$ for the ideal model such that for every $I \subset [n]$ with $|I| < t$,

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

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 $\perp$. Formally,

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 $\mathcal{A}$ 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. Then, every party P_i (for $i > 1$) sends its value x^i to all other parties.*
  3. *Denote the value received by P_i from P_j in the previous round by x^i_j (recall that x^i denotes the value P_i received from P_1 in the first round). Then, P_i outputs x^i if this is the only value that it saw (i.e., if $x^i = x^i_2 = \cdots = x^i_n$). Otherwise, it outputs $\perp$.*

     We note that if P_i did not receive any value in the first round, then it always outputs $\perp$.

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

Proposition 3.1 *Protocol* 1 *solves the broadcast with 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 $x^j = x'$. On the other hand, if P_j does not output x^j, then it outputs $\perp$.

2. *Validity:* If P_1 is honest, then all parties receive x in the first round. Therefore, they will only output x or $\perp$.

This completes the proof. ■

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 for $t \geq n/3$):

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

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

It is easy to see that the above strengthening of the agreement requirement results in the definition of weak Byzantine Generals. (The validity and non-triviality requirements combined together are equivalent to the validity requirement of weak Byzantine Generals.) Therefore, there exists no *deterministic* protocol for the case of $t \geq n/3$. For what can be done if one utilizes probabilistic protocols, see the section on recent *related work* in the introduction. Regarding the strengthening of the validity requirement, the resulting definition implies a problem known as "Crusader Agreement". This was shown to be unachievable for any $t \geq n/3$ by Dolev in [11]. 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 a different real model in which the adversary has control over the delivery of messages. We note that because the network is synchronous, the adversary cannot deliver a message late (i.e., no round r message can be sent in round r' for $r' > r$). The only additional capability that it is given is to *block* messages sent between honest parties.

In this section we show that it is possible to realize an ideal broadcast functionality in a universally composable way within the above-described model, 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 result. An important corollary of this result is the existence of universally composable multi-party computation with $t \geq n/3$, in a point-to-point network. This corollary is obtained by applying our secure realization of broadcast with known universally composable multi-party protocols. See Appendix A for an overview of the universal composition framework.

The ideal broadcast functionality is defined in Figure 1.

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

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

- Upon receiving a message $(\textsf{broadcast}, x)$ from P_i, send $(\textsf{broadcast}, P_i, x)$ 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:

Protocol 2 (universally composable broadcast):

- **Input:** *P_i has input* $(\textsf{broadcast}, sid, x)$.[10]
- **The Protocol:**
  1. *P_i sends (sid, x) to all parties.*
  2. *Denote by x^j the value received by P_j in the previous round. Then, every party P_j (for $j \neq i$) sends its value (sid, x^j) to all other parties.*
  3. *Denote the value received by P_j from P_k in the previous round by x_k^j (recall that x^j denotes the value P_j received from P_i in the first round). Then, P_j outputs* $(\textsf{broadcast}, sid, P_i, x^j)$ *if it received* all *the messages x_k^j and this is the only value that it saw (i.e., if it received x_k^j from every P_k and it holds that $x^j = x_2^j = \cdots = x_n^j$). Otherwise, it outputs nothing.*

     We note that if P_j did not receive any value in the first round, then it does not output anything.

We note the difference between Protocol 2 and Protocol 1. Here, in Step 3 of the protocol, the parties check that they received messages from all the other parties. This forces the adversary to deliver all messages. This is needed because without this modification, honest parties receiving different x values may be unable to notify each other (the adversary can simply not deliver any of the messages sent between them). The main result of this section is the following proposition:

Proposition 3.2 *Protocol* 2 *securely realizes $\mathcal{F}_{\text{BC}}$ (in the universally composable framework).*

Proof: Let $\mathcal{A}$ be a real-model adversary attacking Protocol 2. We construct an ideal model adversary $\mathcal{S}$ for $\mathcal{A}$ that interacts with $\mathcal{F}_{\text{BC}}$. We differentiate between two cases: in the first the dealer is corrupted (and thus is controlled by $\mathcal{A}$), and in the second it is honest. Let P_i be the dealer in this execution.

- *Case* 1 – *P_i is corrupt:* In the first round, $\mathcal{A}$ (controlling P_i) sends messages to ℓ of the honest parties for some ℓ; denote these messages by $x_{i_1}, \ldots, x_{i_\ell}$. Simulator $\mathcal{S}$ receives all these messages and then simulates the messages sent by the honest parties in the second round. Furthermore, $\mathcal{S}$ obtains all the messages sent by $\mathcal{A}$ in the second round.

  Now, if there exist j and k ($1 \leq j, k \leq \ell$) such that $x_{i_j} \neq x_{i_k}$, then $\mathcal{S}$ sends nothing to $\mathcal{F}_{\text{BC}}$. Otherwise, let x be the message sent by $\mathcal{A}$. Then, $\mathcal{S}$ sends x to the ideal functionality $\mathcal{F}_{\text{BC}}$. Next, $\mathcal{S}$ defines the set of honest parties to whom to deliver the output $(\textsf{broadcast}, sid, P_i, x)$

[10]We note that by the current formalization of universal composability, all participating parties hold a unique and common session identifier *sid*. We remark that this is included for convenience and is not actually needed for achieving universal composability.

from $\mathcal{F}_{\text{BC}}$. This set of parties is defined to be those to whom $\mathcal{A}$ delivers all the second round messages and whose messages all contain the same x. $\mathcal{S}$ concludes by delivering the messages from the $\mathcal{F}_{\text{BC}}$ functionality to these parties, and only to these parties.

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

We claim that the global output of an ideal execution with $\mathcal{S}$ is *identically distributed* to the global output of a real execution with $\mathcal{A}$. We first deal with the case that P_i is corrupt. If $\mathcal{A}$ sends two different messages in round 1 (i.e., if there exist j and k such that $x_j \neq x_k$), then by the protocol definition, all honest parties will see both x_j and x_k. (Here it is important that parties do not output x unless seeing the round 2 messages of all parties.) Therefore, in a real execution all honest parties will output nothing. This is identical to the case that $\mathcal{S}$ does not send anything to $\mathcal{F}_{\text{BC}}$ in an ideal execution. In contrast, if $\mathcal{A}$ sends the same message x to all honest parties in the first round, then the outputs depend on what $\mathcal{A}$ sends in the second round. Since $\mathcal{S}$ receives all these messages from $\mathcal{A}$, it can see which parties would output x and which parties would output nothing. $\mathcal{S}$ thus delivers the $(\mathsf{broadcast}, ...)$ messages from $\mathcal{F}_{\text{BC}}$ only to the parties which would output x in the real model. We conclude that the output is identical.

In the case that P_i is honest, $\mathcal{A}$ can cause honest parties to output nothing (rather than x) by sending them messages $x' \neq x$ in the second round or by not delivering messages. As above, $\mathcal{S}$ receives all these messages and therefore its delivery of $(\mathsf{broadcast}, ...)$ messages from $\mathcal{F}_{\text{BC}}$ accurately represents exactly what happens in a real execution. ■

We now apply Proposition 3.2 in order to obtain universally composable secure computation without a broadcast channel. Our first corollary relates to the scenario where a majority of the parties are honest, but this majority may be less than 2/3 (i.e., $n/3 \leq t < n/2$). In this scenario, Canetti [6] showed that universally composable protocols exist for any functionality, assuming that the parties interact in a synchronous network with a broadcast channel. Combining this with Proposition 3.2 we have the following:

Corollary 1 *Consider a synchronous point-to-point network where the adversary controls message delivery. Then, for any multi-party ideal functionality $\mathcal{F}$, there exists a* universally composable *protocol* Π *that securely realizes $\mathcal{F}$ in the presence of malicious, static adversaries, and for $t < n/2$ corruptions.*

The next corollary relates to a setting with an honest *minority.* In this setting, universal composability cannot be achieved without somehow augmenting the model [7, 6]. One augmentation that is used is that of a common reference string [7, 9]. Canetti et al. [9] show that in a synchronous network with a *broadcast channel* and a common reference string, it is possible to securely compute any functionality. Therefore, by combining this with Proposition 3.2, we have that:

Corollary 2 *Consider a synchronous point-to-point network where the adversary controls message delivery, and assume that trapdoor permutations exist. Then, for any multi-party ideal functionality $\mathcal{F}$, there exists a* universally composable *protocol* Π *in the common reference string model, that securely realizes $\mathcal{F}$ in the presence of malicious, static adversaries, and for any number of corruptions.*

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. The result of this section is formally stated in the following theorem:

Theorem 3 *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 information-theoretic (resp., computational)* t*-secure computation with unanimous abort and any level of fairness, then* Π' *is a protocol for information-theoretic (resp., computational)* t*-secure computation with abort and no fairness.*

Combining Theorem 3 with known protocols (specifically, [28] and [18][11]), we obtain the following corollaries:

Corollary 4 (information-theoretic security – compilation of [28]): *For any 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.*

We note that this result is optimal in the following sense. Ben-Or et al. [4] showed that there are functions for which there do not exist information-theoretically private protocols when $t \geq n/2$. The definition of a "private" (rather than "secure") protocol, is regarding the behavior of corrupted parties. In a private protocol, corrupted parties follow the protocol specification, but attempt to learn more information than intended (such adversarial behavior is known as passive or semi-honest). Therefore, security with abort implies privacy, and this means that for information-theoretic security, resilience of $t \geq n/2$ is not possible. On the other hand, the result is not optimal regarding fairness. (In particular, the recent [14] achieve an analogous result with complete fairness.)

Corollary 5 (computational security – compilation of [18]): *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 no fairness, for* any t.

We now proceed to prove Theorem 3.

Proof of Theorem 3: Intuitively, we construct a protocol for the point-to-point model from a protocol for the broadcast model, by having the parties in the point-to-point network simulate the broadcast channel. When considering "pure" broadcast (i.e., Byzantine Generals), this is not possible for $t \geq n/3$. However, it suffices to simulate the broadcast channel using a protocol for "broadcast with abort". Recall that in such a protocol, either the correct value is delivered to all parties, or some parties output $\perp$. The idea is to halt the computation in the case that any honest party receives $\perp$ from a broadcast execution. The point at which the computation halts dictates

[11] Both the [28] and [18] protocols have simultaneous termination

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 by parties to notify each other that they have received $\perp$. 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. Likewise, if a party receives $\perp$ in a blank round, then it sends $\perp$ to all parties in the next blank round (it also does not participate in the next broadcast).

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 proceed to 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'. The parties emulate Π according to the following instructions:*

  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.*

  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.*

In order to prove that 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 that is computational (resp., information-theoretic) t-secure 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 computational (resp., information-theoretic) t-secure 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.

As we have mentioned, we first define the adversary $\mathcal{A}$ who attacks Π. Adversary $\mathcal{A}$ internally invokes $\tilde{\mathcal{A}}$ and therefore has internal communication with $\tilde{\mathcal{A}}$ and external communication with the honest parties executing Π.

Adversary $\mathcal{A}$ for Π:

- **Input:** $\mathcal{A}$ receives an input sequence $\{x_i\}_{i\in I}$ and a series of random-tapes $\{r_i\}_{i\in I}$. (Recall that $\tilde{\mathcal{A}}$ controls all 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 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 $\textsc{real}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r})$ be the global output of an execution of Π with adversary $\mathcal{A}$, inputs $\overline{x}$, and random-tapes $\overline{r}$ (i.e., $\overline{r} = (r_1,\ldots,r_n)$ where P_i receives random-tape r_i). (Thus, $\textsc{real}_{\Pi,(\mathcal{A},I)}(\overline{x}) = \{\textsc{real}_{\Pi,(\mathcal{A},I)}(\overline{x},U_{|\overline{r}|})\}$.)
- For any subset $J \subseteq [n]$, denote by $\textsc{real}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r})\,|_J$, the *restriction* of $\textsc{real}_{\Pi,(\mathcal{A},I)}(\overline{x},\overline{r})$ 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 on the parties' inputs $\overline{x}$ and random-tapes $\overline{r}$. We denote by $J = J_{\tilde{\mathcal{A}}}(\overline{x},\overline{r})$ the set of parties who do not output $\perp$ in an execution of $\tilde{\Pi}$ with adversary $\tilde{\mathcal{A}}$ (and where the inputs and random-tapes are $\overline{x}$ and $\overline{r}$). (Notice that $J_{\tilde{\mathcal{A}}}$ is a fixed function depending only on $\overline{x}$ and $\overline{r}$.) We also denote by $J_{\tilde{\mathcal{A}}}(\overline{x})$ a random variable taking values over $J_{\tilde{\mathcal{A}}}(\overline{x},\overline{r})$ for uniformly distributed $\overline{r}$.

We now consider the joint distribution of the outputs of the adversary and the parties in $J_{\tilde{\mathcal{A}}}(\overline{x},\overline{r})$ (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, and for all input and random-tape sequences $\overline{x}$ and $\overline{r}$,

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

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 $\tilde{\mathcal{A}}$ may have very different outputs in $\mathcal{A}$ (and in particular may output their prescribed outputs). Nevertheless, at this stage we are only interested in those parties not outputting $\perp$.

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 $\mathcal{S}$. Recall that $\tilde{S}$ works in an ideal model for secure computation with abort and no fairness, whereas $\mathcal{S}$ works in an ideal model for secure computation with unanimous abort and partial fairness. Further 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: $\tilde{\mathcal{S}}$ receives input series $\{x_i\}_{i\in I}$ and internally invokes $\mathcal{S}$ upon the same inputs $\{x_i\}_{i\in I}$. Then, $\tilde{\mathcal{S}}$ 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 two cases:

1. P_1 *is not corrupted:* in this case, $\mathcal{S}$ concludes at this point, outputting some value.
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. 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 $\tilde{\mathcal{S}}$. Formally, we claim the following: For every set I, every set of inputs $\overline{x}$ and every random-tape r (for $\mathcal{S}$ or $\tilde{\mathcal{S}}$),

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

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

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

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 of IDEAL (resp., REAL). This implies that,

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

(Otherwise, we could distinguish $\text{IDEAL}^{(2)}_{f,(\mathcal{S},I)}(\overline{x})$ from $\text{REAL}_{\Pi,(\mathcal{A},I)}(\overline{x})$ by comparing 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

$$\left\{\text{IDEAL}^{(5)}_{f,(\tilde{\mathcal{S}},I)}(\overline{x})|_{J_{\tilde{\mathcal{S}}}(\overline{x})}\right\} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x})|_{J_{\tilde{\mathcal{A}}}(\overline{x})}\right\}$$

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,

$$\left\{\text{IDEAL}^{(5)}_{f,(\tilde{\mathcal{S}},I)}(\overline{x})\right\} \overset{\text{c}}{\equiv} \left\{\text{REAL}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x})\right\}$$

completing the proof of Lemma 4.1. ■

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

Lemma 4.2 *Let* Π *be a protocol in the broadcast model that is information-theoretic or computational t-secure with unanimous abort and with any level of fairness, 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* $|I| < t$*,*

$$\left\{\text{REAL}_{\tilde{\Pi},(\tilde{\mathcal{A}},I)}(\overline{x})\right\} \equiv \left\{\text{REAL}_{\Pi',(\mathcal{A}',I)}(\overline{x})\right\}$$

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', there is no difference if a party receives $\perp$ in a broadcast with abort execution or in the following blank round. 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 $\tilde{\Pi}$. On the other hand, 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 $r+1$:* Let $\mathcal{P}$ denote the set of honest parties receiving $\perp$ in any of the above simulated "broadcast with abort" executions (i.e., for the broadcast of rounds r and $r+1$). Then, for every $P_i \in \mathcal{P}$, adversary $\tilde{\mathcal{A}}$ sends $\perp$ to P_i 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 exactly who these parties are and can send them $\perp$ in the following blank round. The key point is that in Π' it *makes no difference* if $\perp$ is received 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 3: Let $\mathcal{A}'$ be an adversary attacking Π'. By Lemma 4.2, we have that there exists an adversary $\tilde{\mathcal{A}}$ attacking $\tilde{\Pi}$ such that the output distributions of Π' with $\mathcal{A}'$, and $\tilde{\Pi}$ with $\tilde{\mathcal{A}}$ are identical. Then, by Lemma 4.1, we have that for real-model adversary $\tilde{\mathcal{A}}$ for $\tilde{\Pi}$, there exists an ideal-model simulator $\tilde{\mathcal{S}}$ such that the output distributions of a real execution with $\tilde{\mathcal{A}}$ and an ideal execution (with abort and no fairness) with $\tilde{\mathcal{S}}$ are computationally indistinguishable (or statistically close). We conclude that the output distribution of a real execution of Π' with adversary $\mathcal{A}'$ is computationally indistinguishable (or statistically close) to an ideal execution (with 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. On the other hand, the bandwidth of Π' is the same as that of 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 different messages, then the bandwidth is upper-bound by the length of the longest message times n.) Therefore, the number of bits sent in an execution of Π' is only 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. (This construction assumes the existence of trapdoor permutations.) Furthermore, for any functionality f, there exists a protocol for *information-theoretic* $n/2$-secure computation of f with abort and partial fairness (and without any complexity assumptions).

Outline: We begin by motivating why the strategy used to obtain secure computation with abort and no fairness is not enough here. 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 communication between honest parties. That is, of course, unless this communication need not be broadcast. Now, in the definition of security with abort and partial fairness, once an honest P_1 receives its output, it must be able to give this output to all honest parties. That is, the adversary must not be allowed to disrupt the communication, following the time that an honest P_1 receives its output. This means that using a "broadcast with abort" protocol in the final stage where the remaining parties receive their outputs is problematic.

We solve this problem here by having the parties compute a different functionality. This functionality is such that when P_1 gets its output, it can supply all the other parties with their output directly and without broadcast. On the other hand, P_1 itself should learn nothing of the other parties' outputs. As a first attempt, consider 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 its output $f_i(\overline{x})$. 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 all secure protocols. 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 $h_i \in_R \mathcal{H}$.
- 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 6 *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.*[12] *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 $\perp$.

[12]By Corollaries 4 and 5 in Section 4, such protocols exist for any t assuming the existence of trapdoor permutations. Furthermore, for the case of $t > n/2$, no assumptions are required.

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.) On the other hand, 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 Canetti [5]. This theorem states that we can consider a hybrid model in which an ideal call is used for Stage 1 of the protocol, whereas the other stages are as described above. That is, 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 [5] 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. Notice that in the ideal call of Stage 1, $\mathcal{A}$ receives all the corrupted parties' outputs first and then decides which honest parties receive output (this is because there is no fairness in the computation of Stage 1). We now construct an ideal-model adversary $\mathcal{S}$ who works in an ideal model with abort and partial fairness. We stress that $\mathcal{A}$ works in a hybrid model in which the "ideal model" part has no fairness; whereas, $\mathcal{S}$ works in an ideal model with partial fairness. $\mathcal{S}$ 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.[13] (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 honest, $\mathcal{S}$ 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

[13] $\mathcal{S}$ 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.

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.

The fact that the global output in the hybrid execution with $\mathcal{S}$ is identically distributed to the global output in a real execution with $\mathcal{A}$ is derived from the following observations. 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 6 ■

Acknowledgements

We would like to thank Oded Goldreich for many helpful discussions.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[15] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Byzantine Agreement Secure Against Faulty Majorities From Scratch. To appear in *PODC,* 2002.

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

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

[18] 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 [17].

[19] 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.

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

[21] J. Kilian, A general completeness theorem for two-party games. In Proc.23rd Annual ACM Symposium on the Theory of Computing, pp. 553- 560, New Orleans, Louisiana, 6-8 May 1991.

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

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

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

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

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

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

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

[29] 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 [6]; for more details, see [6]. The framework provides a formal method for defining the security of cryptographic tasks, while ensuring that security is maintained under a general composition operation in which a secure protocol for the task in question is run in a system concurrently with an unbounded number of other arbitrary protocols. This composition operation is called universal composition, and tasks that fulfill the definitions of security in this framework are called universally composable (UC).

As in other general definitions (e.g., [19, 25, 1, 5]), the security requirements of a given task (i.e., the functionality expected from a protocol that carries out the task) are captured via a set of instructions for a "trusted party" that obtains the inputs of the participants and provides them with the desired outputs (in one or more iterations). We call the algorithm run by the trusted party an ideal functionality. Informally, a protocol securely carries out a given task if an adversary can gain no more in an attack on a real execution of the protocol, than from an attack on an ideal process where the parties merely hand their inputs to a trusted party with the appropriate functionality and obtain their outputs from it, without any other interaction. The fact that the adversary gains no more from its attack on a real execution is formalized by saying that the result of a real execution can be *emulated* in the above ideal process. We stress that in a real execution of the protocol, no trusted party exists and the parties interact amongst themselves only.

In order to prove the universal composition theorem, the notion of emulation in this framework is considerably stronger than in previous ones. Traditionally, the model of computation includes the parties running the protocol and an adversary $\mathcal{A}$, that controls the communication channels and potentially corrupts parties. Then, security is formulated by requiring that for any adversary $\mathcal{A}$ attacking a real protocol execution, there should exist an "ideal process adversary" or simulator $\mathcal{S}$, that causes the outputs of the parties in the ideal process to be essentially the same as the outputs of the parties in a real execution. However, in the universally composable framework, an additional adversarial entity called the environment $\mathcal{Z}$ is introduced. This environment generates the inputs to all parties, reads all outputs, and in addition interacts with the adversary in an arbitrary way throughout the computation. (As is hinted by its name, $\mathcal{Z}$ represents the external environment that consists of arbitrary protocol executions that may be running concurrently with the given protocol.) A protocol is said to securely realize a given ideal functionality $\mathcal{F}$ if for any "real-life" adversary $\mathcal{A}$ that interacts with the protocol there exists an "ideal-process adversary" $\mathcal{S}$, such that *no environment* $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol, or with $\mathcal{S}$ and parties that interact with $\mathcal{F}$ in the ideal process. (In a sense, here $\mathcal{Z}$ serves as an "interactive distinguisher" between a run of the protocol and the ideal process with access to $\mathcal{F}$. See [6] 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 [6]: Consider a protocol π that operates in a *hybrid* model of computation where parties can communicate as usual, and in addition have ideal access to (an unbounded number of copies of) some ideal functionality $\mathcal{F}$. (This model is called the $\mathcal{F}$-hybrid model.) Furthermore, let ρ be a protocol that securely realizes $\mathcal{F}$ as sketched above, and let π^ρ be the "composed protocol". That is, π^ρ is identical to π with the exception that each interaction with the ideal functionality $\mathcal{F}$ is replaced with a call to (or an invocation of) an appropriate instance of the protocol ρ. Similarly, ρ-outputs are treated as values provided by the functionality $\mathcal{F}$. The theorem states that in such a case, π and π^ρ have essentially the same input/output behavior. Thus, ρ behaves just like the ideal functionality $\mathcal{F}$, even when composed with an arbitrary protocol π. A special case of this theorem states that if π securely realizes some ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, then π^ρ securely realizes $\mathcal{G}$ from scratch.