Secure Computation without Agreement
(Extended Abstract)*

Shafi Goldwasser and Yehuda Lindell

Department of Computer Science and Applied Math,
Weizmann Institute of Science, Rehovot, Israel.
{shafi,lindell}@wisdom.weizmann.ac.il

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 was 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.

* A full version of this paper can be found on the IACR Cryptology ePrint Archive, Report 2002/040, http://eprint.iacr.org

D. Malkhi (Ed.): DISC 2002, LNCS 2508, pp. 17–32, 2002.

1 Introduction

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

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:* The outputs received by the parties are guaranteed to be 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 output only if honest parties do. (As with the previous item, this is not always achievable and is therefore not always fully required.)

The standard definition today [17,1,24,5] 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 [22]. 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 $O(t)$ [25], and by a probabilistic protocol with expected round complexity $O(1)$ [10];
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 [25,22]. (This public-key infrastructure is assumed to be setup in a trusted preprocessing phase.) We note that an information theoretic analogue also exists [26]. 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 [4,8], 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 [27]

[1] The protocol of [16] 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.

and in the computational setting [16] 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 [16,19,20]. 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 [28,14,17,2].
   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 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 $t \geq n/3$ [25]. Thus, it is also impossible to achieve general 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 $t < n/3$ [23]. This has the following ramifications. On the one hand, in the range of $n/3 \leq t < n/2$, it is *impossible* to obtain general 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 [15], 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 results hold in both the information theoretic and 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 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:

[2] We note that in private communication, Goldreich stated that the requirement in [15] of having all parties abort or all parties receive output was only made in order to simplify the definition.

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 [15] 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. (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, 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 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 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 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 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. (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 [23] 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.

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

[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 [10]. However, there is no known authenticated Byzantine agreement protocol with analogous round complexity.

light on the difficulties in solving the general problem. However, in the case of secure computation this is not the case. Rather, the fact that the lower bounds of Byzantine agreement apply to secure computation is due to marginal issues relating to unanimity regarding the delivery of outputs, and not due to the main issues of security.

1.5 Related Work

Two recent independent results [12,13] study a problem similar to ours, although apparently for different motivation. They construct protocols for weak Byzantine Agreement for the cases of $t \geq n/3$ and then use this to obtain secure computation without the use of a broadcast channel. In short, they achieve secure computation with unanimous abort whereas we achieve secure computation with abort. However, our protocols are significantly more round efficient. See the full version of this paper for a careful and detailed comparison [18].

1.6 Organization

Due to lack of space in this extended abstract, we omit the formal description of the hierarchy of definitions of secure computation outlined in Section 1.4. We also omit the proofs of our constructions. We refer the reader to the full version of our paper for these and other details [18].

2 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 1. (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 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.

2.1 Strengthening Broadcast with Abort

A natural question to ask is whether or not we can strengthen Definition 1 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 $t \geq n/3$. 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 [9]. 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 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-theoretically* (*resp., computationally*) *secure computation with unanimous abort and any level of fairness, then* Π' *is a protocol for information-theoretically* (*resp., computationally*) *secure computation with abort and no fairness. Furthermore,* Π' *tolerates the same number of corruptions as* Π*.*

Combining Theorem 3 with known protocols (specifically, [27] and [16][4]), we obtain the following corollaries:

Corollary 2. (information-theoretic security – compilation of [27]): *For any probabilistic polynomial-time* n*-ary functionality* f*, there exists a protocol in the point-to-point model, for the information-theoretically secure computation of* f *with abort and no fairness, and tolerating any* $t < n/2$ *corruptions.*

Corollary 3. (computational security – compilation of [16]): *For any probabilistic polynomial-time* n*-ary functionality* f*, there exists a protocol in the point-to-point model, for the computationally secure computation of* f *with abort and no fairness, and tolerating* any $t < n$ *corruptions.*

[4] Both the [27] and [16] protocols have simultaneous termination

Due to lack of space in this extended abstract, the proof of Theorem 3 is omitted. Nevertheless, we present the motivation and construction of the protocol compiler.

The Protocol Compiler: 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 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 2 (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 the security of the compiled protocol, we show how to transform an ideal-model simulator for the original protocol into an ideal-model simulator for the compiled protocol. Although intuitively this seems straightforward, the actual proof is quite involved. See the full version for details.

4 Secure Computation with Abort and Partial Fairness

In this section we show that for any functionality f, there exists a protocol for the *computationally* secure computation of f with abort and partial fairness, and tolerating any $t < n$ corruptions. (This construction assumes the existence of trapdoor permutations.) Furthermore, for any functionality f, there exists a protocol for the *information-theoretically* secure computation of f with abort and partial fairness (and without any complexity assumptions), tolerating any $t < n/2$ corruptions.

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 sent over the broadcast channel. 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 the parties compute the following instead of the original functionality f:

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 P_i's output $f_i(\overline{x})$ 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 $h : \{0,1\}^k \to \{0,1\}^k$. 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. Using the above construction, we obtain the following theorem:

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

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 and M. Fischlin. Universally Composable Commitments. In *CRYPTO*, 2001.
7. R. Canetti and H. Krawczyk. Universally Composable Notions of Key-Exchange and Secure Channels. In *EUROCRYPT*, 2002.
8. D. Chaum, C. Crepeau and I. Damgard. Multi-party Unconditionally Secure Protocols. In *20th STOC*, pages 11–19, 1988.
9. D. Dolev. The Byzantine Generals Strike Again. *Journal of Algorithms,* 3(1):14–30, 1982.
10. P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement. *SIAM. J. Computing*, 26(2):873–933, 1997.

11. M. Fischer, N. Lynch, and M. Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. *Distributed Computing*, 1(1):26–39, 1986.
12. M. Fitzi, N. Gisin, U. Maurer and O. Von Rotz. Unconditional Byzantine Agreement and Multi-Party Computation Secure Against Dishonest Minorities from Scratch. To appear in *Eurocrypt 2002*.
13. M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein and A. Smith. Byzantine Agreement Secure Against Faulty Majorities From Scratch. To appear in *PODC,* 2002.
14. Z. Galil, S. Haber and M. Yung. Cryptographic Computation: Secure Fault Tolerant Protocols and the Public Key Model. In *CRYPTO 1987*.
15. O. Goldreich. *Secure Multi-Party Computation*. Manuscript. Preliminary version, 1998. Available from `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.
16. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th STOC,* pages 218–229, 1987. For details see [15].
17. S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90*, Springer-Verlag (LNCS 537), pages 77–93, 1990.
18. S. Goldwasser and Y. Lindell. Secure Computation Without Agreement (full version of this paper). IACR Cryptology ePrint Archive, Report 2002/040, `http://eprint.iacr.org`
19. J.Kilian, Founding Cryptograph on Oblivious Transfer. In *20th STOC*, pages 20–31, 1988.
20. 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.
21. L. Lamport. The weak byzantine generals problem. In *JACM*, Vol. 30, pages 668–676, 1983.
22. L. Lamport, R. Shostack, and M. Pease. The Byzantine generals problem. *ACM Trans. Prog. Lang. and Systems*, 4(3):382–401, 1982.
23. Y. Lindell, A. Lysyanskaya and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *34th STOC*, 2002.
24. S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.
25. M. Pease, R. Shostak and L. Lamport. Reaching agreement in the presence of faults. In *JACM*, Vol. 27, pages 228–234, 1980.
26. B. Pfitzmann and M. Waidner. Information-Theoretic Pseudosignatures and Byzantine Agreement for $t >= n/3$. Technical Report RZ 2882 (#90830), IBM Research, 1996.
27. T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *21st STOC*, pages 73–85, 1989.
28. A. Yao. How to Generate and Exchange Secrets. In *27th FOCS*, pages 162–167, 1986.