Sequential Composition of Protocols Without Simultaneous Termination

Yehuda Lindell*
Dept. of Computer Science
Weizmann Institute of Science
Rehovot 76100, ISRAEL
lindell@wisdom.weizmann.ac.il

Anna Lysyanskaya*
MIT LCS
200 Technology Square
Cambridge, MA 02139 USA
anna@theory.lcs.mit.edu

Tal Rabin
IBM T.J.Watson Research
PO Box 704, Yorktown Heights
NY 10598, USA
talr@watson.ibm.com

ABSTRACT

The question of the composition of protocols is an important and heavily researched one. In this paper we consider the problem of sequential composition of synchronous protocols that do not have simultaneous termination; i.e., the parties do not necessarily conclude a protocol execution in the same round. A problem arises becauses such protocols must begin in synchrony; therefore a second execution cannot follow from the first in a straightforward manner. An important example of a protocol with this property is that of randomized Byzantine Agreement with an expected constant number of rounds (such as the one due to Feldman and Micali). We note that expected constant-round Byzantine Agreement *cannot* have simultaneous termination and thus this (problematic) property is inherent.

Given that the termination of the parties is not simultaneous, a natural question to consider is how to synchronize the parties so that such protocols can be sequentially composed. Furthermore, such a composition should preserve the original running-time of the protocol, i.e. running the protocol ℓ times sequentially should take in the order of ℓ times the running-time of the protocol. In this paper, we present a method for sequentially composing any protocol in which the players do not terminate in the same round, while preserving the original round complexity. An important application of this result is the sequential composition of parallel Byzantine Agreement. Such a composition can be used by parties connected in a point-to-point network to run protocols designed for the broadcast model, while maintaining the original round complexity.

1. INTRODUCTION

The question of the composition of protocols is a fundamental one. After having designed a protocol, it is important to know what happens to it under composition, as in most cases there is a need to execute the protocol more than once. Two crucial issues that arise when composition is considered are protocol security and complexity. That is, does the protocol maintain security under sequential, parallel or concurrent composition, and is the running-time of the basic protocol preserved under these types of composition? The literature contains many papers which have studied these questions for specific and for general protocols [11, 8, 15, 3, 4, 12]. Most of these works relate to the security of the protocol in question under composition. Yet examining the running time of the composed protocol has rarely been looked at (an exception is [1]). This is due to the fact that composing a protocol ℓ times usually yields a running time which is ℓ times the running time of a single invocation of the protocol.

In this paper we study a very natural set of protocols for which the above does not hold. In particular, naive sequential composition does not result in a running time which is ℓ times the running time of a single invocation. The characteristic of these protocols is that the parties do not terminate their execution in concert and there is no way for a given party to know exactly when the others terminate. We refer to such protocols as *staggered* since there is some termination staggering gap. Intuitively, this staggering causes a problem if the protocol requires synchronized initialization because then it is not possible to merely start the second execution of the protocol immediately after the previous one terminates.

The most prominent protocol which belongs to this set of protocols is that of *randomized* Byzantine Agreement (BA). Randomization was introduced by Rabin [14] and Ben-Or [2] in order to beat known lower bounds for Byzantine Agreement. In particular, [14] achieves a protocol that runs in constant expected time, in contrast to the lower bound of Fischer and Lynch [9], which states that in a synchronous model where there are n parties out of which at most $t < n/3$ may become faulty, any *deterministic* protocol requires $t+1$ rounds of communication.

Later, Feldman and Micali [10] presented the first randomized protocol in the information-theoretic setting which solved the BA problem in constant expected time. Yet, the protocols of [14, 10] have the property that the honest parties do not necessarily terminate in the same round. That is, assuming that the first honest party terminates at time T,

*This work was carried out while the first and second authors were visiting IBM Research.

then we are only guaranteed that all other honest parties terminate by time $T + c$. In this case, we call c the *staggering gap* of the protocol. We require this staggering gap to be a fixed value, and not a random variable that is dependent on the execution. It is interesting to note that in this case, staggered termination is inherent. That is, it is impossible to construct a protocol for the Simultaneous BA problem (where all honest parties are required to terminate in the same round), which has *executions* running for less than $t + 1$ rounds [7, 13]. Thus, any randomized BA protocol that has an expected constant number of rounds, must have a staggering gap.

Now let us consider the problem of sequentially composing a randomized BA protocol with a staggering gap in more detail. We use the randomized BA protocol as a representative of the group of protocols with staggered termination, yet the following description is general and applies to any such protocol. The existing protocols for BA in the synchronous model require that the parties start executing the protocol simultaneously, i.e. all at the same time. Under this stipulation the correctness properties are ensured. Now, in a scenario where many executions $\mathrm{BA}_1, \mathrm{BA}_2, \ldots$ of a staggered protocol are run one after another, the parties no longer satisfy the requirement of starting the execution of BA_{i+1} at the same time. This is because the execution of BA_i completes in a staggered manner. Given that the requirement for synchronicity is broken, it is unclear what can be said with respect to the correctness of the protocol.

A straightforward solution to this problem would be to expand each time unit for the execution of a step in BA_{i+1} according to the staggering gap generated by BA_i. Given that the staggering gap of BA_i was c_i, an honest party would need to count $2c_i + 1$ time units in the execution of BA_{i+1} before moving to the next step.[1] Since the staggering gap of the underlying BA protocol is c, the staggering gap for BA_{i+1} is $c_{i+1} = c(2c_i + 1)$. This means that after i invocations and given an initial staggering gap of c, the running time and staggering termination gap blow up to $O(2^{i-1}c^i)$. Therefore, the running time of the ℓ^{th} execution is exponential in ℓ.

In this paper we present a method for sequentially composing a staggered protocol ℓ times while maintaining:

1. A fixed staggering gap which is independent of the number of sequential invocations ℓ,

2. An expected running time that is equal to ℓ times the expected running time of the staggered protocol that we are composing. We concentrate in our discussion on the round complexity; however, our methodology preserves the order of the communication and computational complexity as well.

The main idea behind our method is to insert synchronizing steps between the sequential executions of the protocol. These synchronizing steps are implemented using a Byzantine Agreement protocol.[2] There are two delicate issues which need to be addressed by the solution:

1. The exact time at which to insert the synchronizing step, in order to guarantee that all parties have completed the execution of the previous protocol before beginning the next execution.

2. How to synchronize parties when the synchronizing protocol itself is staggered. This issue arises because we use a BA protocol with an expected constant number of rounds for the synchronization, which is itself necessarily staggered [13, 7]. We note that if the protocol being composed runs for $t+1$ or more rounds, then we could simply use a Simultaneous BA protocol in order to synchronize. This would simplify our protocol and analysis, yet would result in a blowup of $O(t)$ rather than $O(1)$ rounds.[3]

Parallel Composition of BA. The parallel composition of a protocol with a constant expected number of rounds does not necessarily result in a protocol with the same expected number of rounds. For example, if one composes the Feldman-Micali BA protocol [10] n times in parallel, then the expected running time of the composed protocol is $O(\log n)$, and not constant. Ben-Or and El-Yaniv [1] show how to achieve parallel composition of the [10] protocol, such that the composed protocol also runs in constant expected-time. Yet, in their protocol they need to sequentially compose single invocations of the Feldman-Micali protocol, and so they incur a staggering gap expansion (albeit for only a constant number of executions). Thus our solution can be incorporated into the Ben-Or and El-Yaniv protocol in order to achieve parallel composition more efficiently. In addition, by composing our result with the [1] protocol, we obtain a protocol which enables the sequential composition of many parallel BA executions, and where the overall composed protocol maintains constant expected running time for each parallel execution. This result is important as it enables the efficient simulation of any protocol designed using a broadcast channel, within a point-to-point network. We discuss this additional result in more detail in Section 6.

Relation to Asynchronous Protocols. One possible approach to solving the above-described problem is to simply switch to an asynchronous model whenever this problem arises. In such a model, parties need not begin a protocol execution at the same time and thus sequential composition becomes straightforward. However, this approach is problematic for the following reasons. Firstly, it may be that the protocol being composed does not have an analogous asynchronous version (or if it does, it may be that the asynchronous counterpart is less efficient). Secondly, the sequential composition may be a subprotocol within a larger

[1] Why the expansion is $2c_i + 1$ and not merely c_i is a delicate point which is derived from the proof of correctness of the protocol with expanded steps. The details of this issue appear in the proof of Lemma 3.1.

[2] One should not confuse the synchronizing protocol which is a BA, with the staggered protocol that we are trying to compose (which might also happen to be a BA protocol).

[3] We note that our solution is actually relevant for any protocol whose expected number of rounds is some function less than $t + 1$, and not just for protocols with a constant expected number of rounds.

synchronous protocol. For example, the classic use of Byzantine Agreement is for simulating a broadcast channel. When the protocol using the broadcast channel is synchronous, it is not possible to replace the broadcast by an asynchronous Byzantine Agreement protocol. Finally, if the asynchronous protocol is also staggered (for example the asynchronous BA protocols which run in constant expected time are staggered [6, 5]), then the sequential composition will also experience expansion in the running time, independent of the adversary's actions. Thus, efficiency also becomes an issue in this setting.

2. DEFINITIONS

In this section we present the computational model and the definition for Byzantine Agreement. We also clarify what is meant by the sequential composition of protocols.

We consider a setting involving n parties, denoted $P_1, \dots, P_n$, that are connected via authenticated point-to-point communication channels in a synchronous network. In the basic point-to-point communication model, each party is formally modelled by an interactive Turing machine with $n-1$ pairs of communication tapes. The communication of the network proceeds in *synchronized* rounds, where each round consists of a *send* phase followed by a *receive* phase. In the *send* phase of each round, the parties write messages onto their output tapes, and in the *receive* phase, the parties read the contents of their input tapes. An adversary is called *rushing* if, in every round, it sees the messages of the honest players before computing its own.

In this model, an adversary controls a subset of the parties and the corruption strategy depends on the adversary's view (i.e., the adversary is adaptive). Since the adversary controls these parties, it receives their entire views and determines the messages that they send. In particular, these messages need not be according to the protocol execution, but rather can be computed by the adversary as an arbitrary function of its view. Throughout the paper, we always assume that the adversary can corrupt at most $t < n/3$ of the parties.

Definition 1. (Byzantine Agreement): *Let $P_1, \dots, P_n$ be n parties, with associated inputs $x_1, \dots, x_n$, and let there be an adversary who may corrupt up to $\lfloor (n-1)/3 \rfloor$ of the parties. Then, a protocol solves the* Byzantine Agreement problem *if the following two properties hold:*

1. Agreement: *All honest parties output the same value.*
2. Validity: *If more than $2/3$ of the parties have the same input value x and follow the protocol specification, then all honest parties output x.*

In this paper, we are interested in the execution of ℓ protocols sequentially. In particular, we require that no honest party should begin the next protocol until the previous protocol has concluded. Formally, we say that a party P_i *begins* an execution of a protocol when it sends its first message of that execution. We note that P_i may receive messages belonging to an execution prior to the time that it sends its first message (and thus before it begins). On the other hand, we say that a protocol has *concluded*, if all honest parties have terminated.

Definition 2. (sequential composition): *Let $A_1, \dots, A_\ell$ be ℓ protocols. We say that $A_1, \dots, A_\ell$ are run* sequentially*, if for every i, no honest party begins A_i until A_{i-1} has concluded.*

3. SEQUENTIAL COMPOSITION

In this section we present a protocol for the sequential composition of a staggered protocol ℓ times, so that the expected running time of the composition is ℓ times the expected running time of the underlying protocol. Recall that a staggered protocol is one for which the parties do not necessarily terminate in the same round. Rather, there is a fixed value c, known as the staggering gap, and all the parties are guaranteed to finish within an interval of c rounds.

3.1 Motivation for the Protocol

As we have shown in the Introduction, the main problem with a naive composition of staggered protocols is that both the running time of the protocol and its staggering gap grow exponentially. Therefore, our protocol works by introducing synchronizing steps. The aim of this synchronization is to ensure that the parties start the staggered protocols at almost the same time, i.e. without too much of a gap. A simplistic way of doing this would be to define fixed points on a time line and have the protocol executions begin only at these points. However, the problem with such an approach is that there is no guarantee that previous executions of the protocol have terminated before a new execution begins. In such a case, the executions are not sequential. This problem arises due to the fact that only the *expected* round-complexity of the underlying protocol is constant. Therefore, some executions may run for a very long time.[4]

Our solution to this problem is to still define fixed points on a time line. However, instead of starting a new invocation of the staggered protocol whenever one of these points is reached, the parties first *vote* on whether or not they are ready to start the next run. All that is required from this vote is that if *all* the honest parties are still running the previous execution, then the result of the vote is negative and a new execution will not begin. If the result of a vote is negative, then the parties wait until the next fixed point and cast a revote. On the other hand, if the result of a vote is positive, then this implies that at least one honest party completed the previous run of the staggered protocol. In this case, the parties begin the next execution immediately after the vote concludes. (As we will see, it is enough that one party has concluded the previous run in order to ensure that the executions are sequential.) As can be imagined, this vote is carried out by running a Byzantine Agreement protocol. We call these vote executions *Synchronizing Byzantine*

[4] One can always show that, asymptotically in n, the probability that such a protocol runs for n steps is negligible (and can thus be ignored). However, we must then allocate n steps for each execution, rather than a constant. This is overly wasteful and in such a case deterministic protocols that run for $t+1$ rounds could be used instead. Furthermore, an error (albeit negligible) is introduced. Our solution, on the other hand, introduces no error.

Agreements, denoted SBA (not to be confused with Simultaneous Byzantine Agreement [13, 7]). As described in the Introduction, we cannot use a deterministic BA protocol for the synchronization, as this would increase the running time of the composition by a factor of $t+1$. Therefore, we use an expected randomized Byzantine Agreement protocol that concludes in constant expected time instead. Unfortunately, this means that the synchronizing protocols are also staggered.

3.2 A Protocol with Staggered Initiation

As we have mentioned, the Synchronizing Byzantine Agreement (SBA) protocols are themselves staggered. Furthermore, we wish to use the SBA protocols to synchronize the executions of a second protocol that requires a simultaneous initialization. (In our general setting the second protocol is the one which we are ultimately trying to compose sequentially.) Therefore, we need to show how to modify a protocol which has a requirement for simultaneous initialization into one that can start in a staggered manner. This modified protocol can then be run immediately after the SBA. Specifically, our goal is the following: given a protocol Π and a constant c, modify the protocol Π so that it remains correct even when the parties start its execution within c steps of each other. Intuitively, this is not difficult because the gap c is known and thus we can *expand* each time unit by some fixed amount (dependent on c), thereby returning to a synchronous setting. The protocol Π can have a staggering gap as well, which we denote by d, though d might equal 0. This expanded protocol is called $\mathsf{Expand}\Pi$ and is presented in the proof of the following lemma.

LEMMA 3.1. *Let Π be a protocol and let d be its (fixed) staggering gap. Then there exists a transformation of Π into a protocol $\mathsf{Expand}\Pi$, such that if all honest parties begin $\mathsf{Expand}\Pi$ within c steps of each other (for some constant c), then $\mathsf{Expand}\Pi$ satisfies the following:*

1. *The outputs of $\mathsf{Expand}\Pi$ and Π are the same. Formally speaking, for every adversary $\mathcal{A}$ attacking $\mathsf{Expand}\Pi$ there exists a rushing adversary $\mathcal{A}'$ attacking Π, such that for every vector of inputs $\overline{x}$, the output of all parties from an execution of $\mathsf{Expand}\Pi$ with $\mathcal{A}$ (where the parties' input to $\mathsf{Expand}\Pi$ is $\overline{x}$), is identically distributed to the output of all parties from an execution of Π with $\mathcal{A}'$ (where the parties' input is $\overline{x}$).*
2. *The running time of $\mathsf{Expand}\Pi$ (expected and worst) equals $2c+1$ times the running time of Π.*
3. *The staggering gap of protocol $\mathsf{Expand}\Pi$ equals $c + d(2c+1)$.*

PROOF. Loosely speaking, $\mathsf{Expand}\Pi$ works by defining intervals of size $(2c+1)$. Then, for every round of Π, each party considers messages received $\pm c$ rounds from when it sends its own Π-message (thus the intervals are of size $2c+1$). The key point is that since the honest parties begin within c rounds of each other, it is guaranteed that all honest parties consider the messages sent by other honest parties. Furthermore, it is guaranteed that there is a distinct separation between rounds of Π (and thus the effect is of a fully synchronous execution).

We now formally describe the protocol $\mathsf{Expand}\Pi$. Assume that protocol Π's steps are labeled $p_1, p_2, \ldots$. We further assume without loss of generality that messages sent in Step p_{i+1} are completely determined by the messages which were received in Step p_i[5]. We denote the time units of $\mathsf{Expand}\Pi$ by $T_1, T_2, \ldots$. The protocol description is as follows:

PROTOCOL 1. (Protocol $\mathsf{Expand}\Pi$): *Let T_k denote the round that party P_k starts $\mathsf{Expand}\Pi$.*

1. *At time T_k, party P_k sends the messages from Step p_1 of Π, and at time $T_k + (2c+1)$ it sends the messages relating to Step p_2 of Π.*

   In general, P_k sends the messages of Step p_i of Π at time $T_k + i(2c+1)$.

   The messages associated with Step p_i of Π are labeled $(\mathsf{Expand}\Pi, p_i)$.
2. *Whenever a party receives a message with the labeling $(\mathsf{Expand}\Pi, p_i)$ it stores the message along with the time T_j of protocol $\mathsf{Expand}\Pi$ in which it was received.*
3. *In order to determine its Step p_{i+1} message, party P_k examines the messages labeled with $(\mathsf{Expand}\Pi, p_i)$ which it received and stored between the steps $T_k + i(2c+1) - c$ and $T_k + i(2c+1) + c$. This is an interval of size 2c+1 steps centered around the time at which it sent its own p_i message. It computes the messages of Step p_{i+1} by applying the decision process of Π to these messages.*
4. *Each party outputs the output defined in protocol Π.*

Note that the times that messages are sent and the intervals in which received messages are considered, are different for every honest party and are dependent on the local times T_k in which the parties start $\mathsf{Expand}\Pi$.

We now prove that $\mathsf{Expand}\Pi$ satisfies the requirements of Lemma 3.1. We begin by showing that requirement (1) is fulfilled. In order to do this, we first show that all honest parties consider the messages of all other honest parties, as one would hope. Furthermore, no honest party sends its p_{i+1} messages before all p_i messages have been sent. That is,

CLAIM 3.2. *Let P_k be an honest party and let T_k denote the time at which it (locally) starts the execution. Then in the interval $T_k + i(2c+1) - c$ to $T_k + i(2c+1) + c$, which P_k considers in order to compute its p_{i+1} message,* all *honest parties send their p_i message, and only that message. This holds for all i.*

PROOF. This is shown via a simple calculation. An honest P_l sends its p_i message at time $T_l + i(2c+1)$. First we need

[5]This can be achieved by having party P_k send *itself* its complete view from the previous steps.

to show that P_l sends its p_i message in the specified interval. Recall that all honest parties begin ExpandΠ within at most c steps of each other and thus $T_k - c \leq T_l \leq T_k + c$. Plugging this into the time specification of P_l for message p_i we have that it falls inside P_k's time interval. That is, for every i, $T_k + i(2c+1) - c \leq T_l + i(2c+1) \leq T_k + i(2c+1) + c$. A similar calculation shows that for all $j \neq i$ the time for sending message p_j falls outside of the p_i interval. This completes the proof of the claim. ■

Note that the above claim does not prohibit the adversary from sending a message of Step p_i to P_k after an honest party P_l has sent a message of Step p_{i+1}, and by that making the message dependent on P_l's message. But rather it implies that any such delayed message would not fall in the interval considered by P_k, and thus will anyway be disregarded in the execution of the protocol.

We are now ready to prove requirement (1). Let $\mathcal{A}$ be an adversary for ExpandΠ. Then, we construct a rushing adversary $\mathcal{A}'$ for Π, such that the output distribution of Π is identical to the output distribution of ExpandΠ with adversary $\mathcal{A}$. $\mathcal{A}'$ begins a simulation of the execution of ExpandΠ by invoking a run of the protocol Π. For each round $i \geq 1$ of Π, adversary $\mathcal{A}'$ first receives the honest parties' Step p_i messages. Recall that $\mathcal{A}'$ is rushing and therefore receives the honest parties' messages before sending its own. Then, $\mathcal{A}'$ runs $\mathcal{A}$ for $2c+1$ steps providing the messages sent by the honest parties at the time steps required, as defined in ExpandΠ. Furthermore, throughout these (virtual) steps, it records the Step p_i messages that $\mathcal{A}$ sends to the honest parties. Any other messages are ignored by $\mathcal{A}'$. At the end of the $2c+1$ steps, $\mathcal{A}'$ passes all these recorded messages from $\mathcal{A}$ to the appropriate honest parties participating in Π, thus completing Step p_i.

We claim that the above simulation by $\mathcal{A}'$ is as required. First, notice that an honest party's view of messages received from other honest parties is identical in Π and ExpandΠ. This is because by Claim 3.2, all honest parties send their messages in ExpandΠ in the specified intervals and are therefore considered by other honest parties. On the other hand, trivially all honest parties in Π receive the messages of all other honest parties. Next, notice that if $\mathcal{A}$ only sends its messages in the specified intervals, then $\mathcal{A}'$ can pass these messages at the correct time in Π. On the other hand, if $\mathcal{A}$ sends a message outside of the specified interval, then $\mathcal{A}'$ ignores this and thus the honest parties of Π do not receive the message. However, in ExpandΠ these messages are also ignored. Therefore, an honest party's view of messages received from corrupted parties (that are not ignored) is also identical in Π and ExpandΠ. Finally, we note that $\mathcal{A}$'s view in this simulation is also identical to its view in ExpandΠ. This is because $\mathcal{A}'$ passes $\mathcal{A}$ all the honest parties' messages at the times as defined in ExpandΠ. We have that all parties' views in this simulation by $\mathcal{A}'$ while running an execution of Π, are identical to their views in an execution of ExpandΠ with adversary $\mathcal{A}$.

We now proceed to show that requirement (2) is fulfilled for ExpandΠ. The fact that the worst-case running-time is $(2c+1)$ times the running-time of Π is trivial, since each step of Π takes $(2c+1)$ steps in ExpandΠ. However, the fact that the expected running-time is also $(2c+1)$ times that of Π does not follow from the same argument. This is because the expected running-time of the protocol may depend on the adversary's capabilities. For example, the [10] protocol runs in expected constant-time assuming that the parties' coin-tosses are kept private (otherwise, the adversary can prevent the execution from ever halting). We must therefore show that our modification of Π to ExpandΠ conserves the expected running-time. This is demonstrated by showing that for every adversary $\mathcal{A}$ for ExpandΠ there exists an adversary $\mathcal{A}'$ for Π such that if the expected running-time of ExpandΠ with $\mathcal{A}$ is R rounds, then the expected running-time of Π with $\mathcal{A}'$ is $R/(2c+1)$ rounds. It would then follow that the expected running-time of ExpandΠ is at most $(2c+1)$ times the expected running-time of Π (with respect to any adversary). We note that the adversary $\mathcal{A}'$ required is exactly that demonstrated for requirement (1) above. This is because every $(2c+1)$ rounds of ExpandΠ are perfectly simulated in exactly one round of Π.

We conclude with the fact that the third requirement clearly follows: the staggering gap of ExpandΠ equals the staggering gap of Π multiplied by the time $2c+1$ needed to complete a step in ExpandΠ, plus the initial staggering in the start of ExpandΠ, which is c. Thus the total total staggering gap of ExpandΠ equals $c + d(2c+1)$. This completes the proof of the lemma. ■

3.3 The Protocol

Before presenting the protocol itself, there is one more technicality which must be dealt with. As described in Section 3.1, the protocol works by running SBA protocols at fixed intervals on a time-line. The parties use the SBA to agree on whether or not to begin the next sequential execution. If yes, then they run protocol ExpandΠ (instead of Π itself). The vote on whether or not to execute the next ExpandΠ protocol is such that if at least one honest party has concluded the previous ExpandΠ execution, then the result of the vote may be positive. Therefore, it is possible that some honest parties are still running the previous ExpandΠ when an SBA begins, and nevertheless, the result of the SBA is a positive vote. Carrying this one step further, it is possible that some honest parties are still running the previous ExpandΠ when the SBA concludes, and thus when the new ExpandΠ is supposed to begin. This cannot be allowed to happen since the ExpandΠ executions must be run strictly sequentially (without any overlap).

This problem is solved in the following way. First, notice that if one honest party has concluded ExpandΠ before the SBA begins, then all other honest parties will conclude within the staggering gap of ExpandΠ, which by Lemma 3.1 equals $c + d(2c+1)$ steps. Therefore, as long as we make sure that the SBA *always* runs for at least $c + d(2c+1)$ steps, we are guaranteed that the new ExpandΠ will not begin until the previous one concludes. This condition is easily achieved by stalling any party that concludes the SBA in less than $c + d(2c+1)$ steps. Note that doing so cannot increase the staggering gap because an SBA always starts in a synchronized fashion. We are now ready to present the protocol.

Components of the Protocol:

- *Expanded Staggered Protocols* (ExpandΠ)*:* These are expected constant-round protocols with staggering gap d, that have undergone the transformation from Π as described in Lemma 3.1. The protocol below is used for composing ℓ of these ExpandΠs sequentially. We denote them by Expand$\Pi(1), \ldots,$ Expand$\Pi(\ell)$.

- *Synchronizing Byzantine Agreements* (SBA)*:* These protocols are defined exactly according to the underlying, expected constant-round, Byzantine Agreement protocol. The best known candidate for the SBA protocol is the Feldman-Micali Byzantine agreement [10]. Let c denote the staggering gap of the protocol and assume that this protocol runs for a minimum of $c + d(2c + 1)$ rounds for every player.[6] Finally, denote by k the expected number of rounds until SBA terminates. For simplicity of exposition, we assume that both c and k are constant values (as in the protocol of [10]), although the composition is unaffected when these are arbitrary values.

PROTOCOL 2. (protocol for sequential composition):

- Execution of Synchronizing Byzantine Agreements: *Let T be the current round number. If T is a multiple of $2k$, then the parties run a Synchronizing Byzantine Agreement protocol (denoted* SBA_T*) where their inputs are defined as follows:*
  1. *If Party P_j is not currently running any* ExpandΠ *or previous* SBA, *then it inputs* 1 *into* SBA_T.
     We stress that P_j is said to be currently executing a protocol also in the case that it is scheduled to send the first or last message of the protocol in round T.
  2. *Otherwise (if P_j is currently executing some* ExpandΠ *or previous* SBA*), then P_j inputs* 0 *into* SBA_T.

  In order to ensure that the messages sent by P_j in this protocol are not confused with messages from different (concurrently running) protocols, P_j concatenates the string "SBA_T" *to every message belonging to the* SBA_T *execution.*

- Execution of Staggering Protocols: *Let* SBA_T *be an* SBA *that concluded in the previous round.*
  1. *If Party P_j's output from* SBA_T *equals* 0*, then P_j does not begin a new* ExpandΠ *(but does continue participating in any previous* SBA *or* ExpandΠ *protocols that are still running).*
  2. *If Party P_j's output from* SBA_T *equals* 1 *and this is the i^{th}* SBA *to terminate with* 1*, then P_j begins its execution of* Expand$\Pi(i)$ *in the next round, according to the protocol definition of* ExpandΠ *in Lemma* 3.1 *(Protocol* 1*).*
     As above, P_j concatenates the string "Expand$\Pi(i)$" *to all messages from this protocol.*

- Termination: *Party P_j halts after* Expand$\Pi(\ell)$ *terminates.*

[6] As we have mentioned this can always be achieved by having parties stall until $c + d(2c + 1)$ rounds pass. Furthermore, assuming that c is constant, this does not significantly affect the complexity.

4. PROOF OF CORRECTNESS

In this section, we prove that the effect of running Protocol 2 is the same as the effect of running the Π protocols sequentially. In particular, we show that for every i, the execution of Expand$\Pi(i)$ begins strictly after the execution of Expand$\Pi(i-1)$ concludes. This is enough because by Lemma 3.1, the output of ExpandΠ is the same as the output of Π. Thus the sequential composition of ExpandΠ implies the desired result. We first claim that all the SBA executions in Protocol 2 are correct:

LEMMA 4.1. *Let* $\mathsf{SBA}_1, \mathsf{SBA}_2, \ldots$ *denote the* SBA *protocols in an execution of Protocol* 2*. Then, for every i,* SBA_i *constitutes a* correct *Byzantine Agreement protocol.*

PROOF. First consider a simplified version of Protocol 2, where all the messages sent by the honest parties in ExpandΠ are 0 only. Then, clearly these messages have no effect on the correctness of the SBA executions, and we can consider the SBA executions only. What remains is an execution containing multiple Byzantine Agreement protocols that may be running concurrently (these executions may be concurrent because SBA_i may not conclude before SBA_{i+1} begins). However, as was shown in [12, Proposition 2.1], Byzantine Agreement protocols compose concurrently (for any concurrent scheduling). Therefore, all the SBA_i's are correct.

Now, in the general case, the messages sent in ExpandΠ may be arbitrary (and not only 0). Nevertheless, we claim that they can have no effect on the correctness of the SBA. This is due to the following reason. If an adversary attacking Protocol 2 can cause one of the SBA executions to be incorrect, then we can construct an adversary that successfully attacks concurrent Byzantine Agreement. This adversary simulates all the messages from ExpandΠ while running the SBA executions. It is easy to see that this simulation may be carried out. Therefore, since concurrently composed SBA executions are correct, so are the SBA executions of Protocol 2. ■

We remark that the above lemma holds only with respect to ordinary Byzantine Agreement (in contrast to *authenticated* Byzantine Agreement). In particular, the concurrent composition of authenticated Byzantine Agreement has been shown not to hold whenever $t \geq n/3$ [12].[7]

We now use the fact that each SBA protocol is correct to establish that for every i, protocol Expand$\Pi(i+1)$ begins strictly after Expand$\Pi(i)$ concludes. That is, *no* honest party begins Expand$\Pi(i+1)$ until *all* honest parties have concluded Expand$\Pi(i)$.[8]

[7] Despite the fact that authenticated Byzantine Agreement does not compose concurrently, it is possible to run concurrent executions if unique session-identifiers are allocated to each execution [12]. In the setting we are dealing with here, it is possible to simply set the session-identifier to equal the round number in which the SBA began (this is always the same for all honest parties and is unique for each SBA). Therefore, our result can be extended to authenticated Byzantine Agreement.

[8] It is actually enough to require that the output of Expand$\Pi(i)$ is fully determined before Expand$\Pi(i+1)$ begins. However, the above formulation is simpler and requires little extra cost.

LEMMA 4.2. *In an execution of Protocol 2, for every i, all honest parties conclude the execution of protocol* $\mathsf{Expand}\Pi(i)$ *before any honest party sends a message belonging to the execution of protocol* $\mathsf{Expand}\Pi(i+1)$.

PROOF. Let SBA_T be a synchronizing Byzantine Agreement protocol that terminates with 1 and let $\mathsf{Expand}\Pi(i+1)$ be the $\mathsf{Expand}\Pi$ protocol that begins immediately after SBA_T. There are three possible scenarios regarding the status of $\mathsf{Expand}\Pi(i)$ at the point that SBA_T *begins*:

1. *Case 1* – all honest parties are still running $\mathsf{Expand}\Pi(i)$ when SBA_T begins: In this case, all the honest parties input 0 into SBA_T. By the validity requirement of Byzantine Agreement protocols, this implies that all honest parties output 0 from SBA_T. (Notice that we apply Lemma 4.1 here in our assumption that SBA_T is correct.) By the assumption that SBA_T terminates with output 1, we have that this case cannot occur.
2. *Case 2* – all honest parties have already concluded $\mathsf{Expand}\Pi(i)$ when SBA_T begins: In this case, the lemma holds because $\mathsf{Expand}\Pi(i+1)$ begins after SBA_T terminates (and $\mathsf{Expand}\Pi(i)$ concluded even before SBA_T began).
3. *Case 3* – at time T when SBA_T begins, some honest parties have concluded $\mathsf{Expand}\Pi(i)$ and some honest parties are still executing $\mathsf{Expand}\Pi(i)$: This means that there is at least one honest party that concluded $\mathsf{Expand}\Pi(i)$ by time $T-1$. Recall that the staggering gap of $\mathsf{Expand}\Pi(i)$ equals $c+d(2c+1)$, therefore, all honest parties will conclude $\mathsf{Expand}\Pi(i)$ at time at most $T-1+c+d(2c+1)$. By the assumption that SBA_T runs for a minimum of $c+d(2c+1)$ rounds, we have that all honest parties will conclude by time $T+c+d(2c+1)$ (at the latest). This means that $\mathsf{Expand}\Pi(i+1)$ will start no earlier that $T+c+d(2c+1)+1$, resulting in the fact that $\mathsf{Expand}\Pi(i+1)$ starts after $\mathsf{Expand}\Pi(i)$ has completed.

This completes the proof of the lemma. ∎

Combining Lemma 3.1 with Lemma 4.2, we obtain the following theorem:

THEOREM 3. *Let Π be any (staggered) protocol with a fixed staggering gap. Then, for every adversary $\mathcal{A}$ attacking Protocol 2, there exists a rushing adversary $\mathcal{A}'$ attacking ℓ sequential executions of Π, such that for every set of inputs $\overline{x}$, the output of all parties from an execution of Protocol 2 with $\mathcal{A}$ and input $\overline{x}$, is identically distributed to the output of all parties from ℓ sequential executions of Π with $\mathcal{A}'$ and input $\overline{x}$.*

Thus, the effect of running Protocol 2 is the same as running ℓ sequential executions of Π, as desired. Note that when considering a specific Π, it must be ascertained that Π is secure against rushing adversaries in order for the theorem to be of help. In Section 5 we show that Protocol 2 also obtains optimal round complexity.

5. COMPLEXITY

Our goal is to show that the cost of using our technique for running ℓ staggered protocols sequentially is essentially the sum of the costs of running these protocols individually.

Notation: Let A be a protocol and let $r(A)$ be a random variable denoting the number of rounds that an execution of A takes. $E[r(A)]$ denotes the expected number of rounds.

Let k be the expected number of rounds for the Byzantine agreement protocol used for the SBA component of our protocol (i.e., $E[r(\mathsf{SBA})]=k$). We assume that the Byzantine agreement protocol used for SBA has Properties 5.1 and 5.2 below. For the Feldman-Micali protocol, they follow by inspection; we omit the proofs and refer the reader to [10]. The first property tells us that when all honest parties are in agreement regarding their input, then the SBA protocol always terminates quickly. That is,

PROPERTY 5.1. *There exists a constant $k' \leq k$ such that if all honest players input the same value b into the* SBA, *then it is* guaranteed *to terminate with this value b in k' rounds.*

The next property tells us that for any k-round period in the SBA protocol execution, the SBA concludes in this period with probability at least $1-\epsilon$ (for some constant ϵ).

PROPERTY 5.2. *There exists a constant ϵ such that for all integers $i>0$, $\Pr[r(\mathsf{SBA}) \geq ik \mid r(\mathsf{SBA}) \geq (i-1)k] \leq \epsilon$.*

We note that for the Feldman-Micali protocol, k is constant and it holds that $k' \approx k/3$, while $\epsilon \approx (2/3)^3 = 8/27$. We are now ready to begin our analysis of the round complexity of Protocol 2. Recall that the SBA protocols begin every $2k$ rounds exactly according to a fixed scheduling. We begin by showing that k rounds after a new SBA begins, there is at most one SBA being executed. We note that this holds irrespective of the $\mathsf{Expand}\Pi$ executions.

CLAIM 5.3. *For every integer j, at round $2kj+k$ there is at most one* SBA *that is being executed.*

PROOF. We prove this by induction on j. For $j=0$ this is immediate because only one SBA has been started. Assume that the claim holds for round $2k(j-1)+k$; we now prove it for round $2kj+k$. Consider SBA_{2kj} (i.e., the SBA protocol that begins at round $2kj$). By the inductive hypothesis, there is at most one SBA that is being executed at the time that SBA_{2kj} begins (the hypothesis holds for round $2kj-k$ and therefore trivially for round $2kj$ as well). Denote this SBA by SBA_T. There are three possible cases:

1. *Case 1:* All honest parties complete SBA_T before round $2kj$. In this case, the claim immediately holds because only one SBA is running already by round $2kj$.

2. *Case 2:* All honest parties are running SBA_T in round $2kj$. By the instructions of Protocol 2, in this case all honest parties input 0 into SBA_{2kj}. By Property 5.1, we have that SBA_{2kj} concludes by round $2kj + k' < 2kj + k$. Therefore, only SBA_T can still be running by round $2kj + k$.

3. *Case 3:* Some honest parties are running SBA_T in round $2kj$ and some have already concluded. In this case, SBA_T concludes before round $2kj + c$ where c is the staggering gap of the SBA protocol. Since SBA runs for at least $c+d(2c+1)$ steps, we have that $c \leq k$ and SBA_T concludes before $2kj + k$. Therefore, only protocol SBA_{2kj} can be still running by round $2kj+k$.

This completes the proof of the claim. ■

The above claim tells us that only one SBA protocol is running during the k rounds before any new SBA begins. By applying Property 5.2, we have that this SBA will conclude before the new SBA begins with probability $1-\epsilon$. As will be explained below, SBA executions that begin with no prior SBA still running are of importance. We therefore wish to calculate how many SBA protocols are executed until we obtain one which has this property. In the following claim we show that the probability that i SBA executions go past without success , e.g. $\mathsf{SBA}_{2k(j+1)}$ to $\mathsf{SBA}_{2k(j+i)}$, is at most ϵ^i.

CLAIM 5.4. *Denote by X_j a boolean random variable such that $X_j = 1$ if and only if SBA_{2kj} begins when a previous SBA is still running (i.e., some honest party has not yet concluded this previous SBA execution). Then for every j and for every i,*

$$\Pr\left[\bigcap_{l=1}^{i} X_{j+l} = 1\right] \leq \epsilon^i$$

PROOF. First, note that for every j, $\Pr[X_j = 1] \leq \epsilon$. This follows immediately from Property 5.1 and Claim 5.3. That is, by Claim 5.3 there is at most one SBA that is running by round $2kj-k$. Then, Property 5.1 states that the probability that this SBA continues for k more rounds is at most ϵ. Next, notice that for every j and every l, it also holds that

$$\Pr\left[X_{j+l} = 1 \;\middle|\; \bigcap_{\xi=1}^{l-1} X_{j+\xi} = 1\right] \leq \epsilon$$

This is because Property 5.1 holds irrespective of the inputs and the past. Since

$$\Pr\left[\bigcap_{l=1}^{i} X_{j+l} = 1\right] = \prod_{l=1}^{i} \Pr\left[X_{j+l} = 1 \;\middle|\; \bigcap_{\xi=1}^{l-1} X_{j+\xi} = 1\right]$$

we conclude that $\Pr[\cap_{l=1}^{i} X_{j+l} = 1] \leq \epsilon^i$ as required. ■

Our interest in an SBA that begins when no prior SBA protocols are running is as follows. Consider the case that an $\mathsf{Expand}\Pi$ execution concludes and the parties are now waiting for an SBA execution to terminate with 1 in order to begin a new $\mathsf{Expand}\Pi$ execution. Then, the parties are only *guaranteed* to begin a new $\mathsf{Expand}\Pi$ execution when they all input 1 into an SBA protocol (they may output 1 in other cases, but they also may not). However, this can only occur in the case that a new SBA execution begins when no previous SBA is still running. In this context, the above claim tells us that the probability that the parties have to wait more than $2ki$ rounds for such an SBA is exponentially small in i.

We now proceed to compute the expected delay between $\mathsf{Expand}\Pi$ executions. That is, let d_i denote the number of rounds that elapse between the time that the last player concludes $\mathsf{Expand}\Pi(i)$ and the time that the first player begins $\mathsf{Expand}\Pi(i+1)$. The expected value of d_i is computed in the following claim:

CLAIM 5.5. *Let d_i be as defined above. Then,*

$$E[d_i] \leq 2k + k' + 2k\frac{\epsilon}{(1-\epsilon)^2}$$

PROOF. First, note that at most $2k$ steps pass between the time that $\mathsf{Expand}\Pi(i)$ concludes until the time that the first subsequent SBA begins. Furthermore, $\mathsf{Expand}\Pi(i+1)$ is guaranteed to follow from the first SBA that begins when no previous SBA is still running (this is because no SBA or $\mathsf{Expand}\Pi$ protocol is running and thus all parties input 1 into the new SBA). The number of rounds that this "successful" SBA contributes to the delay is at most k' (by Property 5.1).

We therefore have that the expected delay between $\mathsf{Expand}\Pi(i)$ and $\mathsf{Expand}\Pi(i+1)$ is at most $2k+k'$ plus the expected number of rounds to pass until an SBA begins when no prior SBA protocols are running. Recall that by Claim 5.4, the probability that $2kj$ rounds will pass until such an SBA begins is at most ϵ^j. Thus,

$$\begin{aligned} E[d_i] &\leq 2k + k' + \sum_{j=0}^{\infty} 2kj \cdot \epsilon^j \\ &= 2k + k' + 2k \cdot \frac{\epsilon}{(1-\epsilon)^2} \end{aligned}$$

as required. ■

Applying the above claim, we obtain the following theorem regarding the complexity of Protocol 2:

THEOREM 4. *Let Π be a protocol such that the expected number of rounds in an execution of Π equals K. Furthermore, let P_ℓ denote ℓ sequential executions of Π according to the instructions of Protocol 2. Then, $E[r(P_\ell)] = O(\ell K)$.*

PROOF. By the the linearity of expectation, we have that

$$E[r(P_\ell)] = \sum_{i=1}^{\ell} E[r(\mathsf{Expand}\Pi(i))] + \sum_{i=1}^{\ell} E[d_i]$$

First, by Lemma 3.1, $E[r(\mathsf{Expand}\Pi(i))] = (2c+1)K$, for every i. (Recall that c is the staggering gap of the SBA protocol, which in this case is constant.) Thus, $E[r(\mathsf{Expand}\Pi(i))] = O(K)$.

Next, by Claim 5.5, $E[d_i] \leq 2k + k' + 2k\epsilon/(1-\epsilon)^2$, for every i. Since k, k' and ϵ are all constants, we have that $E[d_i] = O(1)$.

Combining the above, we conclude that $E[r(P_\ell)] = O(\ell K)$, concluding the proof. ■

The main focus of this paper is regarding the round complexity of the sequential composition of a protocol Π with expected number of rounds K. Indeed, we have just shown that the complexity of the composition is optimal. That is, ℓ sequential executions require just $O(\ell K)$ rounds. However, it is of importance to also show that the communication and computational complexity of Protocol 2 is not overly expensive. (By communication complexity, we refer to the number of bits sent and by computational complexity we refer to the local computation of every party.) We now briefly analyze these measures.

First, it is clear that the communication and computational cost of running $\mathsf{Expand}\Pi$ is the same as that of running the original protocol Π. It is also clear, that the expected number of additional SBA executions run is $O(\ell K/2k)$, since an SBA is run every $2k$ steps of Protocol 2. However, we can actually make a stronger claim. Namely, we claim that at any point in time, the additional overhead required is at most two SBA executions. This is implied by Claim 5.3 and ensures us that no party will be involved in many SBA executions at one time. (Such a scenario may be very undesirable.) This completes our analysis.

6. SIMULATING PROTOCOLS DESIGNED FOR THE BROADCAST MODEL

Protocol 2 allows the sequential composition of any protocol that has a staggering gap, and in particular Byzantine Agreement protocols of this type. The prime use of such a composition is for running protocols that are designed for the broadcast model, while interacting in a point-to-point network. In such a protocol, at each step a party broadcasts its message. Therefore, what we really need is the sequential composition of Byzantine *Generals* protocols. In a Byzantine Generals protocol there is a dealer with a given input x. The validity requirement is restated so that if the dealer is honest, then all honest parties output x. The composition of such protocols follows immediately from the above because any Byzantine Agreement protocol can be used to achieve Byzantine Generals (with the cost of only one additional round). Specifically, the dealer first sends its input to all parties. Then, the parties run a Byzantine Agreement protocol, inputting the values that they received in the first round. It is easy to see that the resulting protocol fulfills the requirements. By sequentially composing these Byzantine Generals protocols, we obtain a perfect simulation of a broadcast channel, while using a point-to-point network.

However, another issue that arises in such a simulation is the issue of simultaneous broadcast. In general, when analyzing the round complexity of protocols in the broadcast model, we would like to assume that in each round, every party can broadcast a message to all other parties, as this can lead to a significant reduction in the round complexity of the protocol. Therefore, it would be desirable to allow the *sequential composition* of *parallel executions* of Byzantine Generals protocols. A naive way of doing this would be to simply run the Feldman-Micali protocol [10] in parallel at every step. However, this approach does not work for the following reasons. First, the expected number of rounds for n parallel executions of [10] is not constant, but is rather $O(\log n)$. Secondly, and more seriously, the staggering gap of the parallel protocol is no longer *fixed*. Rather, this too, becomes an *expected value*. In this case, Lemma 3.1 does not apply since it only holds for protocols with a fixed staggering gap. (We note that we do not know how to extend Lemma 3.1 to deal with expected staggering gaps.)

Ben-Or and El-Yaniv [1] presented a solution for the parallel composition of the Feldman-Micali BA protocol while preserving a constant expected round complexity. But in their protocol they also need to sequentially compose a staggering BA protocol (albeit for an expected constant number of times). Furthermore, their method of composition results in a protocol with an *expected* staggering gap. Thus, as above, Lemma 3.1 cannot be directly applied to their protocol. Nevertheless, the sequential composition of BA in their protocol can be substituted with the methodology presented in this paper. The composition of the Ben-Or and El-Yaniv protocol with ours yields a protocol for parallel executions of BA with a constant expected running time and a fixed staggering gap. Furthermore, this protocol is secure against a rushing adversary. Therefore, the resulting protocol satisfies the criteria of Theorem 3 and it can be sequentially composed using Protocol 2. Given the above, we obtain the following corollary:

COROLLARY 6.1. *Let* Prot *be an n-party protocol designed for the synchronous broadcast model, and let $P_1, \cdots, P_n$ be n parties. Then, there exists a protocol* Prot$'$ *for the point-to-point network model such that the following holds: For every adversary $\mathcal{A}'$ attacking* Prot$'$ *there exists an adversary $\mathcal{A}$ attacking* Prot *such that the output of $P_1, \ldots, P_n$ running* Prot *with $\mathcal{A}$ is identically distributed to the output of $P_1, \ldots, P_n$ running* Prot$'$ *with $\mathcal{A}'$. Furthermore, the* expected *number of rounds of* Prot$'$ *is only a* constant *times the round complexity of* Prot *(with simultaneous broadcast in each round).*

The protocol Prot$'$ in the corollary is constructed as follows: The parties run Protocol 2 in order to sequentially compose the protocol of [1], where the parties' inputs into each invocation of [1] are defined by Prot (i.e., the parties use [1] in order to simultaneously broadcast the messages as instructed in Prot). From the above discussion, it is clear that Prot$'$ correctly simulates an execution of Prot that uses a broadcast channel.

We conclude by noting an important special case of the above corollary: there exists a round-preserving transformation of any *secure protocol* for computing a function f in the broadcast model, to a secure protocol that computes f in the point-to-point network model.

7. ACKNOWLEDGMENTS

We thank Cynthia Dwork and Marcos Aguilera for pointing out to us that the lower bound of $t+1$ rounds holds even for *randomized* Simultaneous Byzantine Agreement (and therefore that any expected constant-time BA protocol must be staggered).

8. REFERENCES

[1] M. Ben-Or and R. El-Yaniv. Interactive Consistency in Constant Time. Submitted for publication, 1991.

[2] M. Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *Proceeding 2nd Annual Symposium on Principals of Distributed Computing*, pages 27–30. ACM, 1983.

[3] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

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

[5] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantipole: Practical Asynchronous Byzantine Agreement Using Cryptography . In *Proc. 19th ACM PODC*, pages 123–132. ACM, 2000.

[6] R. Canetti and T. Rabin. Optimal Asynchronous Byzantine Agreement. In *Proc. 25th STOC*, pages 42–51. ACM, 1993.

[7] C. Dwork and Y. Moses. Knowledge and Common Knowledge in a Byzantine Environment: The Case of Crash Failures. In *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 149–170. Morgan Kaufman, 1986.

[8] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proc. 30th STOC*, pages 409–418. ACM, 1998.

[9] M. Fischer and N. Lynch. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183–186, 1982.

[10] P. Feldman and S. Micali. An Optimal Algorithm for Synchronous Byzantine Agreement. *SIAM. J. Computing*, 26(4):873–933, 1997.

[11] O. Goldreich and H. Krawczyk. On the composition of zero-knowledge proof systems. *SIAM. J. Computing*, 25(1):169–192, 1996.

[12] Y. Lindell, A. Lysyanskya, and T. Rabin. On the Composition of Authenticated Byzantine Agreement. To appear in STOC 2002, 2002.

[13] Y. Moses and M. R. Tuttle. Programming Simultaneous Actions Using Common Knowledge. In *Proc. 27th FOCS*, pages 208–221. IEEE, 1986.

[14] M. Rabin. Randomized Byzantine Generals. In *Proceeding 24th Annual Symposium on the Foundations of Computer Science*, pages 403–409. IEEE, 1983.

[15] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Eurocrypt '99*, pages 311–326, 1999. LNCS No. 1592.