On the Composition of Authenticated Byzantine Agreement

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

A fundamental problem of distributed computing is that of simulating a (secure) broadcast channel, within the setting of a point-to-point network. This problem is known as Byzantine Agreement and has been the focus of much research. Lamport et al. showed that in order to achieve Byzantine Agreement in the standard model, more than 2/3 of the participating parties must be honest. They further showed that by augmenting the network with a public-key infrastructure, it is possible to obtain secure protocols for any number of faulty parties. This augmented problem is called "authenticated Byzantine Agreement".

In this paper we consider the question of concurrent, parallel and sequential composition of authenticated Byzantine Agreement protocols. We present surprising impossibility results showing that:

1. Authenticated Byzantine Agreement cannot be composed in parallel or concurrently (even twice), if 1/3 or more of the parties are faulty.
2. Deterministic authenticated Byzantine Agreement protocols that run for r rounds and tolerate 1/3 or more faulty parties, can only be composed sequentially less than $2r$ times.

In contrast, we present randomized protocols for authenticated Byzantine Agreement that compose sequentially for any polynomial number of times. We exhibit two such protocols: The first protocol tolerates corruptions of up to 1/2 of the parties, while In the first protocol, the number of faulty parties may be any number less than 1/2. On the other hand, the second protocol can tolerate any number of faulty parties, but is limited to the case that the overall number of parties is $O(\log k)$, where k is a security parameter. Finally, we show that when the model is further augmented so that unique and common session identifiers are assigned to each concurrent session, then any polynomial number of authenticated Byzantine agreement protocols can be concurrently executed, while tolerating any number of faulty parties.

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

STOC'02, May 19-21, 2002, Montreal, Quebec, Canada.
.

1. INTRODUCTION

The Byzantine Generals (Byzantine Agreement[1]) problem is one of the most researched areas in distributed computing. Numerous variations of the problem have been considered under different communication models, and both positive results, i.e. protocols, and negative results, i.e. impossibility and lower bounds on efficiency and resources, have been established. The reason for this vast interest is the fact that the Byzantine Generals problem is the algorithmic implementation of a broadcast channel within a point-to-point network. In addition to its importance as a stand-alone primitive, broadcast is a key tool in the design of secure protocols for multiparty computation.

Despite the importance of this basic functionality and the vast amount of research that has been directed towards it, our understanding of the algorithmic issues is far from complete. As is evident from our results, there are still key questions that have not yet been addressed. In this paper, we provide solutions to some of these questions.

The problem of Byzantine Generals is (informally) defined as follows: There are n parties, one of which is the General who holds an input x. In addition, there is an adversary who controls up to t of the parties and can arbitrarily deviate from the designated protocol specification. The (honest) parties need to agree on a common value. Furthermore, if the General is not faulty, then this common value must be his original input x.

Pease *et al.* [15, 13] provided a solution to the Byzantine Generals problem in the standard model, i.e. the information-theoretic model with point-to-point communication lines (and no setup assumptions). For their solution, the number of faulty parties, t, must be less than $n/3$. Furthermore, they complemented this result by showing that the requirement for $t < n/3$ is in fact inherent. That is, no protocol which solves the Byzantine Generals problem in the standard model can tolerate a third or more faulty parties.

The above bound on the number of faulty parties in the standard model is a severe limitation. It is therefore of great importance to introduce a different (and realistic) model in which it is possible to achieve higher fault tolerance. One possibility involves augmenting the standard model such that messages sent can be authenticated. By authentication, we mean the ability to ascertain that a message was in fact sent by a specific party, even when not directly received from that party. This can be achieved using a trusted preprocessing phase in which a public-key infrastructure for digital signatures (e.g. [18, 11]) is set up. (We note that this

[1] These two problems are essentially equivalent.

requires that the adversary be computationally bounded. However, there exist preprocessing phases which do not require any computational assumptions; see [16].) Indeed, Pease *et al.* [15, 13] use such an augmentation and obtain a protocol for the Byzantine Generals problem which can tolerate *any* number of faulty parties (this is very dramatic considering the limitation to 1/3 faulty in the standard model). The Byzantine Generals problem in this model is referred to as *authenticated* Byzantine Generals.

A common use of Byzantine Generals is to substitute a broadcast channel. Therefore, it is clear that the settings in which we would want and need to run it, involve many invocations of the Byzantine Generals protocol. The question of whether these protocols remain secure when executed concurrently, in parallel or sequentially is thus an important one. However, existing work on this problem (in both the standard and authenticated models) focused on the security and correctness of protocols in a single execution only.

It is easy to see that the unauthenticated protocol of Pease *et al.* [15], and other protocols in the standard model, do compose concurrently (and hence in parallel and sequentially). However, this is not the case with respect to *authenticated* Byzantine Generals. The first to notice that composition in this model is problematic were Gong, Lincoln and Rushby [12], who also suggest methods for overcoming the problem. Our work shows that these suggestions and any others are futile; in fact composition in this model is impossible (as long as 1/3 or more of the parties are faulty). (We note that by composition, we refer to stateless composition; see Section 2.3 for a formal discussion.)

Our Results. Our first theorem, stated below, shows that authenticated Byzantine Generals protocols, both deterministic and randomized, cannot be composed in parallel (and thus concurrently). This is a surprising and powerful statement with respect to the issue of enhancing the standard model by the addition of authentication. The theorem shows that this enhancement *does not* provide the ability to overcome the impossibility result when composition is required. That is, if there is a need for parallel composition, then the number of faulty players cannot be $n/3$ or more, and hence the authenticated model provides no advantage over the standard model.

THEOREM 1. *No protocol for authenticated Byzantine Agreement that composes in parallel (even twice) can tolerate $n/3$ or more faulty parties.*

Regarding the question of sequential composition, we show different results. We first prove another (weaker) lower bound for *deterministic* protocols:

THEOREM 2. *Let Π be a* deterministic *protocol for authenticated Byzantine Agreement that terminates within r rounds of communication. Then, Π can be sequentially composed at most $2r-1$ times.*

In contrast, for *randomized* protocols we obtain positive results and present a protocol which can be composed sequentially (any polynomial number of times), and which tolerates $t < n/2$ faulty parties. The protocol which we present is based on a protocol of Fitzi and Maurer [8] that tolerates $t < n/2$ faulty parties, and is in the standard model augmented with an ideal three-party broadcast primitive. We show that this primitive can be replaced by an authenticated protocol for three parties that can be composed sequentially (and the resulting protocol also composes sequentially). Thus, we prove:

THEOREM 3. *Assume that there exists a signature scheme that is existentially secure against chosen message attacks. Then, there exists a randomized protocol for authenticated Byzantine Generals with a bounded number of rounds, that tolerates $t < n/2$ faulty parties and composes sequentially any polynomial number of times.*

We also present a randomized Byzantine Generals protocol that tolerates *any* number of faulty parties, and composes sequentially any polynomial number of times. However, the number of messages sent in this protocol is exponential in the number of parties. Therefore, it can only be used when the overall number of parties is logarithmic in the security parameter of the signature scheme.

On the Use of Unique Session Identifiers. As will be apparent from the proofs of the lower bounds (Theorems 1 and 2), what prevents agreement in this setting is the fact that honest parties cannot tell in which execution of the protocol a given message was authenticated. This allows the adversary to "borrow" messages from one execution to another, and by that attack the system. In Section 5, we show that if we further augment the authenticate model so that unique and common indices are assigned to each execution, then security under many concurrent executions can be achieved (for any number of faulty parties).

Thus, on the one hand, our results strengthen the common belief that session identifiers are necessary for achieving authenticated Byzantine Generals. On the other hand, we show that such identifiers *cannot be generated within the system.* Typical suggestions for generating session identifiers in practice include having the General choose one, or having the parties exchange random strings and set the identifier to be the concatenation of all these strings. However, Theorem 1 rules out all such solutions (notice that just coming up with a common identifier involves reaching agreement). Rather, one must assume the existence of some *trusted external means* for coming up with unique and common indices. This seems to be a very difficult, if not impossible, assumption to realize in many natural settings.

A natural question to ask here relates to the fact that unique and common session identifiers are anyway needed in order to carry out concurrent executions. In particular, parties need to be able to allocate messages to protocol executions, and this requires a way of distinguishing executions from each other. Indeed, global session identifiers solve this problem. However, it also suffices for each party to allocate *local* identifiers for itself. That is, when a party begins a new execution, it chooses a unique identifier *sid* and informs all parties to concatenate *sid* to any message they send him within this execution. It is then guaranteed that any message sent by an honest party to another honest party will be directed to the execution it belongs to. We thus conclude that for the purposes of carrying out concurrent executions, global identifiers are not needed.

Implications for Secure Multiparty Computations. As we have stated above, one important use for Byzantine Generals protocols is to substitute the broadcast channel in a multiparty protocol. In fact, most known solutions for multiparty computations assume a broadcast channel, claiming that it can be substituted by a Byzantine Generals protocol without any complications. Our results therefore imply that multiparty protocols that rely on authenticated Byzantine Generals to replace the broadcast channel, cannot be composed in parallel or concurrently.

Another important implication of our result is due to the fact that any secure protocol for solving *general* multiparty tasks can be used to solve Byzantine Generals. Therefore, none of these protocols can be composed in parallel or concurrently, unless more than 2/3 of the parties are honest or a physical broadcast channel is available.

Our Work vs. Composition of Secure Multiparty Protocols. There has been much work on the topic of protocol composition in the context of multiparty computation [1, 14, 2, 5, 3]. Much of this work has focused on zero-knowledge and concurrent zero-knowledge protocols [10, 6, 17, 4]. For example, Goldreich and Krawczyk [10] show that there exist protocols that are zero-knowledge when executed stand-alone, and yet do not compose in parallel (even twice). However, protocols that compose do exist (see, for example, Goldreich [9] and references therein). In contrast, we show that it is impossible to obtain *any protocol* that will compose twice in parallel.

2. DEFINITIONS

2.1 Computational Model

We consider a setting involving n parties, $P_1, \ldots, P_n$, that interact in a *synchronous* point-to-point network. In such a network, each pair of parties is directly connected, and it is assumed that the adversary cannot modify messages sent between honest parties. In this setting, each party is formally modeled 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.

This paper refers to the *authenticated* model, where some type of *trusted preprocessing phase* is assumed. This is modeled by all parties also having an additional `setup-tape` that is generated during the preprocessing phase. Typically, in such a preprocessing phase, a public-key infrastructure of signature keys is generated. That is, each party receives its own secret signing key, and in addition, public verification keys associated with all other parties. (This enables parties to use the signature scheme to authenticate messages that they receive, and is thus the source of the name "authenticated".) However, we stress that our lower bound holds for all preprocessing phases (even those that cannot be efficiently generated).

In this model, a t-adversary is a party that controls $t < n$ of the parties $P_1, \ldots, P_n$, where 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. We note that our impossibility results hold even against a static adversary (for whom the set of faulty parties is fixed before the execution begins).

Note that our protocols for authenticated Byzantine Agreement that compose sequentially rely on the security of signature schemes, and thus assume probabilistic polynomial-time adversaries only. On the other hand, our impossibility results hold for adversaries (and honest parties) whose running time is of any complexity. In fact, the adversary that we construct to prove our lower bounds is of the same complexity as the *honest* parties.

2.2 Byzantine Generals/Agreement

The existing literature defines two related problems: Byzantine Generals and Byzantine Agreement. In the first problem, there is one designated party, the General, who wishes to broadcast its value to all the other parties. In the second problem, each party has an input and the parties wish to agree on a value, with a validity condition that if a majority of honest parties begin with the same value, then they must terminate with that value. These problems are equivalent in the sense that any protocol solving one can be used to construct a protocol solving the other, while tolerating the same number of faulty parties. We relax the standard requirements on protocols for the above Byzantine problems in that we allow a protocol to fail with probability that is negligible in some security parameter. This relaxation is needed for the case of authenticated Byzantine protocols where signature schemes are used (and can always be forged with some negligible probability). Formally,

Definition 1. (Byzantine Generals): *Let $P_1, \ldots, P_{n-1}$ and $G = P_n$ be n parties and let G be the designated party with input x. In addition there is an adversary who may corrupt up to t of the parties including the special party G. A protocol solves the Byzantine Generals problem if the following two properties hold (except with negligible probability):*

1. Agreement: *All honest parties output the same value.*

2. Validity: *If G is honest, then all honest parties output x.*

We denote such a protocol by $BG_{n,t}$.

In the setting of Byzantine Agreement it is not straightforward to formulate the validity property. Intuitively, it should capture that if enough honest parties begin with the same input value then they will output that value. By "honest," we mean the parties that follow the prescribed protocol exactly, ignoring the issue that the first step of the party might be to change its local input.

Definition 2. (Byzantine Agreement): *Let $P_1, \ldots, P_n$ be n parties, with associated inputs $x_1, \ldots, x_n$. In addition there is an adversary who may corrupt up to t of the parties. Then, a protocol solves the Byzantine Agreement problem if the following two properties hold (except with negligible probability):*

1. Agreement: *All honest parties output the same value.*

2. Validity: *If $\max(n-t, \lfloor n/2 \rfloor + 1)$ of the parties have the same input value x and follow the protocol specification, then all honest parties output x.*

We note that for the information-theoretic setting, the validity requirement is usually stated so that it must hold only when more than *two thirds* of the parties have the same input value, because in the information-theoretic setting, $n - t > 2n/3$.

Authenticated Byzantine Protocols: In the model for authenticated Byzantine Generals/Agreement, some trusted preprocessing phase is run before any executions begin. In this phase, a trusted party distributes keys to every participating party. Formally,

DEFINITION 3. (Authenticated Byzantine Generals and Agreement): *A protocol for authenticated Byzantine Generals/Agreement is a Byzantine Generals/Agreement protocol with the following augmentation:*

- *Each party has an additional* setup-tape.
- *Prior to any protocol execution, an ideal (trusted) party chooses a series of strings $s_1, \ldots, s_n$ according to some distribution, and sets party P_i's* setup-tape *to equal s_i (for every $i = 1, \ldots, n$).*

Following the above preprocessing stage, the protocol is run in the standard communication model for Byzantine Generals/Agreement protocols.

As we have mentioned, a natural example of such a preprocessing phase is one where the strings $s_1, \ldots, s_n$ constitute a public-key infrastructure. That is, the trusted party chooses key-pairs $(pk_1, sk_1), \ldots, (pk_n, sk_n)$ from a secure signature scheme, and sets the contents of party P_i's tape to equal $s_i = (pk_1, \ldots, pk_{i-1}, sk_i, pk_{i+1}, \ldots, pk_n)$. That is, all parties are given their own signing key and the verification keys of all the other parties.

We remark that the above-defined preprocessing phase is very strong. First, it is assumed that it is run completely by a trusted party. Furthermore, there is no computational bound on the power of the trusted party generating the keys. Nevertheless, our impossibility results hold even for such a preprocessing phase.

2.3 Composition of Protocols

This paper deals with the security of authenticated Byzantine Agreement protocols, when the protocol is executed many times (rather than just once). We define the composition of protocols to be *stateless*. This means that the honest parties act upon their view in a single execution only. In particular, this means that the honest parties do not store in memory their views from previous executions or coordinate between different executions occurring at the current time. Furthermore, in stateless composition, there is no unique session identifier that is common to all participating parties. (See the Introduction for a discussion on session identifiers and their role.) We note that although the parties are stateless, the adversary is allowed to maliciously coordinate between executions and record its view from previous executions. Formally, composition is captured by the following process:

DEFINITION 4. (sequential and parallel composition): *Let $P_1, \ldots, P_n$ be parties for an authenticated Byzantine Generals/Agreement protocol Π. Let $I \subseteq [n]$ be an index set such that for every $i \in I$, the adversary $\mathcal{A}$ controls the party P_i. Over time, indices are added to I as the adversary chooses to corrupt additional parties, with the restriction that $|I| \leq t$. Then, the* sequential (resp., parallel) composition *of Π involves the following process:*

- *Run the preprocessing phase associated with Π and obtain the strings $s_1, \ldots, s_n$. Then, for every j, set the* setup-tape *of P_j to equal s_j.*
- *Repeat the following process a polynomial number of times sequentially (resp., in parallel).*
  1. *The adversary $\mathcal{A}$ chooses an input vector $x_1, \ldots, x_n$.*
  2. *Fix the input tape of every honest P_j to be x_j and the random-tape to be a uniformly (and independently) chosen random string.*
  3. *Invoke all parties for an execution of Π (using the strings generated in the preprocessing phase above). The execution is such that for $i \in I$, the messages sent by party P_i are determined by $\mathcal{A}$ (who also sees P_i's view). On the other hand, all other parties follow the instructions as defined in Π.*

We stress that the preprocessing phase is executed only once and all executions use the strings distributed in this phase. Furthermore, we note that Definition 4 implies that all honest parties are oblivious of the other executions that have taken place (or that are taking place in parallel). This is implicit in the fact that in each execution the parties are invoked with no additional state information, beyond the contents of their input, random and key tapes. On the other hand, the adversary $\mathcal{A}$ can coordinate between the executions, and its view at any given time includes all the messages received in all other executions.[2]

Before proceeding, we show that any Byzantine Generals (or Agreement) protocol *in the standard model* composes concurrently.

PROPOSITION 2.1. *Any protocol Π for Byzantine Generals (or Agreement) in the* standard model*, remains secure under concurrent composition.*

PROOF: We reduce the security of Π under concurrent composition to its security for a single execution. Assume by contradiction that there exists an adversary $\mathcal{A}$ who runs N concurrent executions of Π, such that with non-negligible probability, in one of the executions the outputs of the parties are not according to the requirements of the Byzantine Generals. We construct an adversary $\mathcal{A}'$ who internally incorporates $\mathcal{A}$ and attacks a single execution of Π. Intuitively, $\mathcal{A}'$ simulates all executions apart from the one in which $\mathcal{A}$ succeeds in its attack. Formally, $\mathcal{A}'$ begins by choosing an index $i \in_R \{1, \ldots, N\}$. Then, for all but the i^{th} execution of the protocol, $\mathcal{A}'$ plays the roles of the honest parties in an interaction with $\mathcal{A}$ (this simulation is internal to $\mathcal{A}'$). On the other hand, for the i^{th} execution, $\mathcal{A}'$ externally interacts with the honest parties and passes messages between them and $\mathcal{A}$ (which it runs internally). The key point in the proof is that the honest parties hold no secret information (and do not coordinate between executions). Therefore, the simulation of the concurrent setting by $\mathcal{A}'$ for $\mathcal{A}$ is *perfect*. Thus, with probability $1/N$, the i^{th} execution is the one in which $\mathcal{A}$ succeeds. However, this means that $\mathcal{A}'$ succeeds in breaking the protocol for a single execution (where $\mathcal{A}'$'s success probability equals $1/N$ times the success probability of $\mathcal{A}$.) This contradicts the stand-alone security of Π. ■

[2]The analogous definition for the composition of *unauthenticated* Byzantine Generals/Agreement is derived from Definition 4 by removing the reference to the preprocessing stage and setup-tapes.

3. IMPOSSIBILITY RESULTS

In this section we present two impossibility results regarding the composition of authenticated Byzantine Agreement protocols. Recall that we are concerned with *stateless* composition. First, we show that it is impossible to construct an authenticated Byzantine Agreement protocol that composes in parallel (or concurrently), and is secure when $n/3$ or more parties are faulty. This result is analogous to the Fischer *et al.* [7] lower bound for Byzantine Agreement in the standard model (i.e., without authentication). We stress that our result does not merely show that authenticated Byzantine Agreement protocols do not necessarily compose; rather, we show that one *cannot* construct protocols that will compose. Since there exist protocols for unauthenticated Byzantine Agreement that are resilient for any $t < n/3$ faulty parties and compose concurrently, this shows that the advantage gained by the preprocessing step in authenticated Byzantine Agreement protocols is lost when composition is required.

Next, we show a lower bound on the number of rounds required for *deterministic* authenticated Byzantine Agreement that composes sequentially. (Note that the impossibility of *parallel* composition holds even for randomized protocols.) We show that if an authenticated Byzantine Agreement protocol that tolerates $n/3$ or more faulty parties is to compose sequentially r times, then there are executions in which it runs for more than $r/2$ rounds. Thus, the number of rounds in the protocol is linear in the number of times it is to compose. This rules out any practical protocol that will compose for a (large) polynomial number of times.

Intuition. Let us first provide some intuition into why the added power of the preprocessing step in authenticated Byzantine Agreement does not help when composition is required. (Recall that in the stand-alone setting, there exist authenticated Byzantine Agreement protocols that tolerate any number of faulty parties. On the other hand, under parallel composition, more than $2n/3$ parties must be honest.) An instructive step is to first see how authenticated Byzantine Agreement protocols typically utilize the preprocessing step, in order to increase fault tolerance. A public-key infrastructure for signature schemes is used and this helps in achieving agreement for the following reason. Consider three parties A, B and C participating in a standard (unauthenticated) Byzantine Agreement protocol. Furthermore, assume that during the execution A claims to B that C sent it some message x. Then, B cannot differentiate between the case that C actually sent x to A, and the case that C did not send this value and A is faulty. Thus, B cannot be sure that A really received x from C. Indeed, such a model has been called the "oral message" model, in contrast to the "signed message" model of authenticated Byzantine Agreement [13]. On the other hand, the use of signature schemes helps to overcome this exact problem: If C had *signed* the value x and sent this signature to A, then A could forward the signature to B. Since A cannot forge C's signature, this would then constitute a proof that C indeed sent x to A. Therefore, utilizing the unforgeability property of signatures, it is possible to achieve Byzantine Agreement for any number of faulty parties.

However, the above intuition holds only in a setting where a single execution of the agreement protocol takes place. Specifically, if a number of executions were to take place, then A may send B a value x along with C's signature on x, yet B would still not know whether C signed x in *this* execution, or in a different (concurrent or previous) execution. Thus, the mere fact that A produces C's signature on a value does not provide proof that C signed *this* value in *this* execution. As we will see in the proof, this is enough to render the public-key infrastructure useless under some types of composition.

We remark that it is possible to achieve concurrent composition, using state in the form of unique and common session identifiers. However, as we have mentioned, there are many scenarios where this does not seem to be achievable (and many others where it is undesirable).

THEOREM 1. *No protocol for authenticated Byzantine Agreement that composes in parallel (even twice) can tolerate $n/3$ or more faulty parties.*

PROOF: The proof of Theorem 1 is based on some of the ideas used by Fischer et al. [7] in their proof that no unauthenticated Byzantine Agreement protocol can tolerate $n/3$ or more faulty parties. We begin by proving the following lemma:

LEMMA 3.1. *There exists no protocol for authenticated Byzantine Agreement for three parties, that composes in parallel (even twice) and can tolerate one faulty party.*

PROOF: Assume, by contradiction, that there exists a protocol Π that solves the Byzantine Agreement problem for three parties A, B and C, where one may be faulty. Furthermore, Π remains secure even when composed in parallel twice. Exactly as in the proof of Fischer et al. [7], we define a hexagonal system S that intertwines two independent copies of Π. That is, let A_1, B_1, C_1 and A_2, B_2 and C_2 be independent copies of the three parties participating in Π. By independent copies, we mean that A_1 and A_2 are the same party A with the same key tape, that runs in two different parallel executions of Π, as defined in Definition 4. The system S is defined by connecting party A_1 to C_2 and B_1 (rather than to C_1 and B_1); party B_1 to A_1 and C_1; party C_1 to B_1 and A_2; and so on, as in Figure 1.

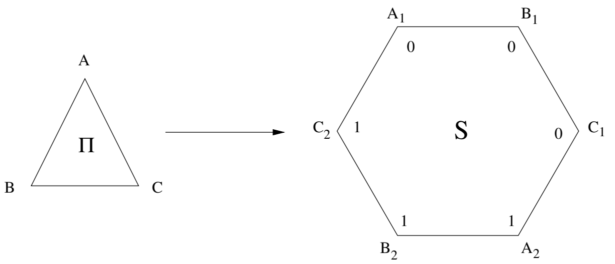

Figure 1: Combining two copies of Π in a hexagonal system S.

In the system S, parties A_1, B_1, and C_1 have input 0; while parties A_2, B_2 and C_2 have input 1. Note that within S, all parties follow the instructions of Π exactly. We stress that S is *not* a Byzantine Agreement setting (where the parties are joined in a complete graph on three nodes), and therefore the definitions of Byzantine Agreement tell us nothing directly of what the parties' outputs should be. However, S *is* a well-defined system and this implies that the parties have well-defined output distributions. The proof proceeds by showing that if Π is a correct Byzantine Agreement protocol, then we arrive at a contradiction regarding the output distribution in S. We begin by showing that B_1 and C_1 output 0 in S. We denote by $\mathsf{rounds}(\Pi)$ the upper bound on the number of rounds of Π (when run in a Byzantine Agreement setting).

CLAIM 3.2. *Except with negligible probability, parties B_1 and C_1 halt within* $\mathsf{rounds}(\Pi)$ *steps and output* 0 *in the system S.*

PROOF: We prove this claim by showing that there exists a faulty party (or adversary) $\mathcal{A}$ who participates in two parallel copies of Π and simulates the system S, with respect to B_1 and C_1's view. The faulty party $\mathcal{A}$ (and the other honest parties participating in the parallel execution) work within a Byzantine Agreement setting where there are well-defined requirements on their output distribution. Therefore, by analyzing their output in this parallel execution setting, we are able to make claims regarding their output in the system S.

Let A_1, B_1 and C_1 be parties running an execution of Π, denoted Π_1, where B_1 and C_1 both have input 0. Furthermore, let A_2, B_2 and C_2 be running a parallel execution of Π, denoted Π_2, where B_2 and C_2 both have input 1. Recall that B_1 and B_2 are independent copies of the party B with the same key tape (as defined in Definition 4); likewise for C_1 and C_2.

Now, let $\mathcal{A}$ be an adversary who controls both A_1 in Π_1 and A_2 in Π_2 (recall that the faulty party can coordinate between the different executions). Party $\mathcal{A}$'s strategy is to maliciously generate an execution in which B_1's and C_1's view in Π_1 is *identical* to their view in S. $\mathcal{A}$ achieves this by redirecting edges of the two parallel triangles (representing the parallel execution), so that the overall system has the same behavior as S; see Figure 2.

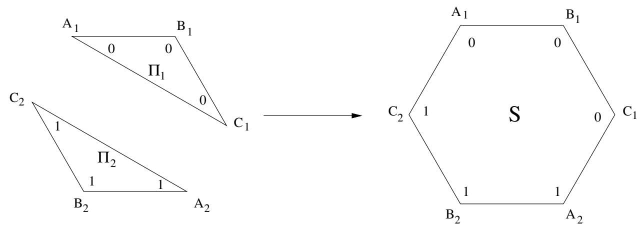

Figure 2: Redirecting edges of Π_1 and Π_2 to make a hexagon.

Specifically, the (A_1, C_1) and (A_2, C_2) edges of Π_1 and Π_2 respectively are removed, and the (A_1, C_2) and (A_2, C_1) edges of S are added in their place. $\mathcal{A}$ is able to make such a modification because it only involves redirecting messages to and from parties that it controls (i.e., A_1 and A_2).

Before proceeding, we present the following notation: let $\mathsf{msg}_i(A_1, B_1)$ denote the message sent from A_1 to B_1 in the i^{th} round of the protocol execution. We now formally show how the adversary $\mathcal{A}$ works. $\mathcal{A}$ invokes parties A_1 and A_2, upon inputs 0 and 1 respectively. We stress that A_1 and A_2 follow the instructions of protocol Π exactly. However, $\mathcal{A}$ provides them with their incoming messages and sends their outgoing messages for them. The only malicious behavior of $\mathcal{A}$ is in the redirection of messages to and from A_1 and A_2. A full description of $\mathcal{A}$'s code is as follows (we recommend the reader to refer to Figure 2 in order to clarify the following):

1. *Send outgoing messages of round i:* $\mathcal{A}$ obtains messages $\mathsf{msg}_i(A_1, B_1)$ and $\mathsf{msg}_i(A_1, C_1)$ from A_1 in Π_1, and messages $\mathsf{msg}_i(A_2, B_2)$ and $\mathsf{msg}_i(A_2, C_2)$ from A_2 in Π_2 (these are the round i messages sent by A_1 and A_2 to the other parties; as we have mentioned, A_1 and A_2 compute these messages according to the protocol definition and based on their view).
   - In Π_1, $\mathcal{A}$ sends B_1 the message $\mathsf{msg}_i(A_1, B_1)$ and sends C_1 the message $\mathsf{msg}_i(A_2, C_2)$ (and thus the (A_1, C_1) directed edge is replaced by the directed edge (A_2, C_1)).
   - In Π_2, $\mathcal{A}$ sends B_2 the message $\mathsf{msg}_i(A_2, B_2)$ and sends C_2 the message $\mathsf{msg}_i(A_1, C_1)$ (and thus the (A_2, C_2) directed edge is replaced by the directed edge (A_1, C_2)).
2. *Obtain incoming messages from round i:* $\mathcal{A}$ receives messages $\mathsf{msg}_i(B_1, A_1)$ and $\mathsf{msg}_i(C_1, A_1)$ from B_1 and C_1 in round i of Π_1, and messages $\mathsf{msg}_i(B_2, A_2)$ and $\mathsf{msg}_i(C_2, A_2)$ from B_2 and C_2 in round i of Π_2.
   - $\mathcal{A}$ passes A_1 in Π_1 the messages $\mathsf{msg}_i(B_1, A_1)$ and $\mathsf{msg}_i(C_2, A_2)$ (and thus the (C_1, A_1) directed edge is replaced by the directed edge (C_2, A_1)).
   - $\mathcal{A}$ passes A_2 in Π_2 the messages $\mathsf{msg}_i(B_2, A_2)$ and $\mathsf{msg}_i(C_1, A_1)$ (and thus the (C_2, A_2) directed edge is replaced by the directed edge (C_1, A_2)).

We now claim that B_1 and C_1's view in Π_1 is identical to B_1 and C_1's view in S.[3] This holds because in the parallel execution of Π_1 and Π_2, all parties follow the protocol definition (including A_1 and A_2). The same is true in the system S, except that party A_1 is connected to B_1 and C_2 instead of to B_1 and C_1. Likewise, A_2 is connected to B_2 and C_1 instead of to B_2 and C_2. However, by the definition of $\mathcal{A}$, the messages seen by all parties in the parallel execution of Π_1 and Π_2 are exactly the same as the messages seen by the parties in S (e.g., the messages seen by C_1 in Π_1 are those sent by B_1 and A_2, exactly as in S). Therefore, the views of B_1 and C_1 in the parallel execution maliciously controlled by $\mathcal{A}$, are *identical* to their views in S.[4]

By the assumption that Π is a correct Byzantine Agreement protocol that composes twice in parallel, we have that, except with negligible probability, in Π_1 both B_1 and C_1 halt within $\mathsf{rounds}(\Pi)$ steps and output 0. The fact that they both output 0 is derived from the fact that B_1 and C_1 are an honest majority with the same input value 0. Therefore, they must output 0 in the face of *any* adversarial A_1; in particular this holds with respect to the specific adversary $\mathcal{A}$ described above. Since the views of B_1 and C_1 in S are identical to their views in Π_1, we conclude that in the system S, they also halt within $\mathsf{rounds}(\Pi)$ steps and out-

[3] In fact, the views of *all* the parties in the parallel execution with $\mathcal{A}$ are identical to their view in the system S. However, in order to obtain Claim 3.2, we need only analyze the views of B_1 and C_1.

[4] We note the crucial difference between this proof and that of Fischer et al. [7]: the faulty party $\mathcal{A}$ is able to simulate the entire $A_1 - C_2 - B_2 - A_2$ segment of the hexagon system S by itself. Thus, in a *single* execution of Π with B_1 and C_1, party $\mathcal{A}$ can simulate the hexagon. Here, due to the fact that the parties B_2 and C_2 have secret information that $\mathcal{A}$ does not have access to, $\mathcal{A}$ is unable to simulate their behavior itself. Rather, $\mathcal{A}$ needs to redirect messages from the parallel execution of Π_2 in order to complete the hexagon.

put 0 (except with negligible probability). This completes the proof of the claim. ■

Using analogous arguments, we obtain the following two claims:

CLAIM 3.3. *Except with negligible probability, parties A_2 and B_2 halt within* $\mathsf{rounds}(\Pi)$ *steps and output* 1 *in the system S.*

In order to prove this claim, the faulty party is $\mathcal{C}$ and it works in a similar way to $\mathcal{A}$ in the proof of Claim 3.2 above. (The only difference is regarding the edges that are redirected.)

CLAIM 3.4. *Except with negligible probability, parties A_2 and C_1 halt within* $\mathsf{rounds}(\Pi)$ *steps and output the same value in the system S.*

Similarly, this claim is proven by taking the faulty party as $\mathcal{B}$ who follows a similar strategy to $\mathcal{A}$ in the proof of Claim 3.2 above.

Combining Claims 3.2, 3.3 and 3.4 we obtain a contradiction. This is because, on the one hand C_1 must output 0 in S (Claim 3.2), and A_2 must output 1 in S (Claim 3.3). On the other hand, by Claim 3.4, parties A_2 and C_1 must output the same value. This concludes the proof of the lemma. ■

Theorem 1 is derived from Lemma 3.1 in the standard way [15, 13] by showing that if there exists a protocol that is correct for any $n \geq 3$ and $n/3$ faulty parties, then one can construct a protocol for 3 parties that can tolerate one faulty party. This is in contradiction to Lemma 3.1, and thus Theorem 1 is implied. ■

The following corollary, referring to concurrent composition, is immediately derived from the fact that parallel composition (where the scheduling of the messages is fixed and synchronized) is merely a special case of concurrent composition (where the adversary controls the scheduling).

COROLLARY 1. *No protocol for authenticated Byzantine Agreement that composes concurrently* (*even twice*) *can tolerate $n/3$ or more faulty parties.*

Sequential Composition of Deterministic Protocols. We now show that there is a significant limitation on *deterministic* Byzantine Agreement protocols that compose *sequentially*. Specifically, any protocol which terminates within r rounds can only be composed sequentially for at most $2r-1$ times. The lower bound is derived by showing that for any deterministic protocol Π, r rounds of the hexagonal system S (see Figure 1) can be simulated in $2r$ sequential executions of Π. As we have seen in the proof of Theorem 1, the ability to simulate S results in a contradiction to the correctness of the Byzantine Agreement protocol Π. However, a contradiction is only derived if the system S halts. Nevertheless, since Π terminates within r rounds, the system S also halts within r rounds. We conclude that the protocol Π can be sequentially composed at most $2r-1$ times.

We remark that in actuality, one can prove a more general statement that says that for *any* deterministic protocol, r rounds of 2 parallel executions of the protocol can be perfectly simulated in $2r$ sequential executions of the same protocol. (More generally, r rounds of k parallel executions of a protocol can be simulated in $k \cdot r$ sequential executions.) Thus, essentially, the deterministic sequential lower bound is derived by reducing it to the parallel composition case of Theorem 1. That is,

THEOREM 2. *Let Π be a* deterministic *protocol for authenticated Byzantine Agreement that concludes after r rounds of communication. Then, Π can be sequential composed at most $2r-1$ times.*

4. SEQUENTIALLY COMPOSABLE RANDOMIZED PROTOCOLS

In this section we present two results. The first one is a protocol which tolerates any $t < n/2$ faulty parties and has polynomial communication complexity (i.e., bandwidth). The second one is a protocol that can tolerate any number of faulty parties but is exponential in the number of participating parties.

The building block for both of the above protocols will be a randomized (sequentially composable) protocol, $\mathrm{ABG}_{3,1}$, for authenticated Byzantine Generals between 3 parties and tolerating one faulty party. Recall that $\mathrm{ABG}_{n,t}$ denotes an authenticated Byzantine Generals protocol for n parties that tolerates up to t faults. We first present the protocol $\mathrm{ABG}_{3,1}$ and then show how it can be used to achieve the above-described results.

4.1 Sequentially Composable $\mathrm{ABG}_{3,1}$

For this protocol we assume three parties: the general, G, and the recipients P_1, P_2. The General has an input value x. According to Definition 1, parties P_1 and P_2 need to output the same value x', and, if G is not faulty, then $x' = x$. As is evident from the proofs of the impossibility, what hinders a solution is that faulty parties can *import* messages from previous executions, and there is no means to distinguish between those and the current messages. Thus, if some freshness could be introduced in the signatures, then this would foil the adversary's actions. Yet, agreeing on such freshness would put us in a circular problem. Nevertheless, the case of three parties is different: here there are only two parties who need to receive each signature. Furthermore, it turns out that it suffices if the parties who are *receiving* a signature can jointly agree on a fresh string. Fortunately, *two* parties can easily agree on a new fresh value: they simply exchange messages and set the fresh string to equal the concatenation of the exchanged values. Now, in the protocol which follows for three parties, we require that whenever a party signs a message, it uses freshness generated by the two remaining parties. We note that in the protocol, only the General, G, signs a message, and therefore only it needs a public key. The protocol is described in Figure 3. For simplicity, we assume that the signature scheme is defined such that $\sigma_{pk}(z)$ also contains the value z.

As we will wish to incorporate Protocol 3 into a protocol with n parties we state a broader claim for the composition than for a simple three party setting.

LEMMA 4.1. *Assume that the signature scheme σ is existentially secure against adaptive chosen message attacks. Then, Protocol* 3 *is a secure protocol for $ABG_{3,1}$ that can be composed sequentially within a system of n parties, in which t may become faulty, for any $t < n$.*

Protocol 3: Authenticated Byzantine Generals for Three Parties

Public Input:	Security parameter 1^k
	Public key pk associated with G
Private Input of G:	Secret key sk corresponding to pk
	Value x

1. P_1 and P_2 agree on a random label ℓ, as follows:
   (a) P_i chooses a random k-bit string u_i and sends it to P_j
   (b) Set $\ell = u_1 \circ u_2$, where $\circ$ denotes concatenation.
2. P_1 and P_2 both send ℓ to G.
3. Let ℓ_i denote the message that G received from P_i in the previous round. G forms $m = \sigma_{sk}(x, \ell_1, \ell_2)$ and sends m to P_1 and P_2.
4. Denote by m_i the message received by P_i in the previous step from G. P_i checks whether the message m_i is a valid signature on the label ℓ (and another, possibly different, label). If so, P_i forwards m_i to P_j.
5. P_i denotes by m'_j the message that it received from P_j in the previous step. P_i outputs according to the following rules based on the examination of m_i and m'_j:
   If all valid signatures on messages containing the label ℓ are for the same value x, then output x.
   Otherwise, output a default value.

Figure 3: $\mathbf{ABG}_{3,1}$

Proof: We prove the theorem by contradiction. Assume that a series of $\mathrm{ABG}_{3,1}$ protocols are run sequentially, such that in some (or all) of them, the adversary succeeded in foiling agreement with non-negligible probability. We will show that in such a case, we can construct a forger $\mathcal{F}$ for the signature scheme who succeeds with non-negligible probability. This will then be in contradiction to the security of the signature scheme.

As there are n parties and the adversary can control up to t of them, there may be executions where two or three of the parties are corrupted. However, in such a case, agreement holds vacuously. On the other hand, any execution in which all three parties are honest must be correct. Therefore, agreement can only be foiled in the case that exactly one participating party is corrupted.

We first claim that when $\mathcal{A}$ plays the General in an execution, it cannot foil the agreement. This is because P_1 and P_2's views of the messages sent by $\mathcal{A}$ (playing G) are identical. Furthermore, their decision making process based on their view is deterministic. Therefore, they must output the same value. We stress that this is irrespective of how many executions have passed (and is also not dependent at all on the security of the signature scheme being used).

Thus, it must be the case that the foiled execution is one where the general is an honest party. As we have mentioned, we build a forger $\mathcal{F}$ for the signature scheme σ who uses $\mathcal{A}$. The forger $\mathcal{F}$ receives as input a public verification-key pk, and access to a signing oracle associated with this key. $\mathcal{F}$ begins by choosing at random one of the parties, say P_j, and associating the verification-key pk with this party. Intuitively, with probability $1/n$, this is the party who plays the general when $\mathcal{A}$ foils the agreement. For all other parties, the forger $\mathcal{F}$ chooses a key pair, for which it knows both the signing and verification keys. Then, $\mathcal{F}$ gives the adversary $\mathcal{A}$ the key pairs for all the initially corrupted parties. $\mathcal{F}$ now invokes $\mathcal{A}$ and simulates the roles of the honest parties in the sequential executions of Protocol 3, with $\mathcal{A}$ as the adversary. In particular, $\mathcal{F}$ works as follows:

- In all executions where the recipient/s P_1 and/or P_2 are not corrupted, $\mathcal{F}$ plays their role, following the protocol exactly as specified. This is straightforward as the recipients do not use signing keys during such an execution.
- In all executions where the general is some uncorrupted party $P_l \neq P_j$, the forger $\mathcal{F}$ plays the role of P_l, following the protocol and using the signing-key which it associated with P_l initially.
- In all executions where the general is the uncorrupted P_j, the forger $\mathcal{F}$ plays the role of P_j following the protocol. However, in this case, $\mathcal{F}$ does not have the associated signing-key. Nevertheless, it does have access to the signing oracle associated with pk (which is P_j's public verification-key). Therefore, $\mathcal{F}$ executes these signatures by accessing its oracle. In particular, for labels ℓ_1, ℓ_2 that it receives during the simulation, it queries the signature oracle for $\sigma_{pk}(x, \ell_1, \ell_2)$.
- *Corruptions:* If at any point, $\mathcal{A}$ corrupts a party $P_l \neq P_j$, then $\mathcal{F}$ hands $\mathcal{A}$ the signing-key that is associated with P_l (this is the only secret information that P_l has). On the other hand, if at any point $\mathcal{A}$ corrupts P_j, then $\mathcal{F}$ aborts (and does not succeed in forging).

Throughout the above-described simulation, $\mathcal{F}$ monitors each execution and waits for an execution in which exactly one party is corrupt and the agreement is foiled. If no such execution occurs, then $\mathcal{F}$ aborts. Otherwise, in the first foiled execution, $\mathcal{F}$ checks if the uncorrupted P_j is the general in this execution. If not, then $\mathcal{F}$ aborts (without succeeding in generating a forgery). Otherwise, we have an execution in which P_j is the general and agreement is foiled. In such a case, $\mathcal{F}$ succeeds in generating a forgery as follows.

As we have mentioned, agreement can only be foiled if exactly one party is faulty. Since by assumption P_j is not corrupted, we have that one of the recipients P_1 or P_2 are corrupted; without loss of generality, let P_1 be the corrupted party. (We note that $\mathcal{F}$ plays the roles of both honest parties P_j and P_2 in the simulation.) Now, since the agreement was foiled, we know that P_2 does not output P_j's input value x, which means that it defaulted in Step 5. This can only happen if P_2 received two valid signatures on the label ℓ which it sent P_j in this execution. Now, P_2 clearly received a correct signature m on P_j's input using the label ℓ from P_j itself. (In fact, by the simulation, this signature is generated by $\mathcal{F}$ accessing its signature oracle.) However, in addition, P_2 must have received a valid signature m' from P_1, where m' constitutes P_j's signature on a string that contains label ℓ and a different message x'. With overwhelming probability the label ℓ did not appear in any previous execution, because P_2 is honest and chooses its portion of the label at random. Thus, previously in the simulation, the signing oracle was never queried with a string containing ℓ. Furthermore, by the assumption that $x' \neq x$, the oracle query by $\mathcal{F}$ in this execution was *different* to the string upon which m' is a signature. We conclude that m' is a valid signature on a message, and that $\mathcal{F}$ did not query the signing oracle with this message. Therefore, $\mathcal{F}$ outputs m' and this is a successful forgery.

It remains to analyze the probability that $\mathcal{F}$ succeeds in this forgery. First, it is easy to see that when $\mathcal{F}$ does not abort, the simulation of the sequential executions is *perfect*, and that $\mathcal{A}$'s view in this simulation is identical to a real execution. Furthermore, the probability that P_j is the identity of the (uncorrupted) general in the first foiled agreement equals $1/n$ exactly. The fact that P_j is chosen ahead of time makes no difference because the simulation is perfect. Therefore, the choice of P_j by $\mathcal{F}$ does not make any difference to the behavior of $\mathcal{A}$. We conclude that $\mathcal{F}$ succeeds in forging with probability $1/n$ times the probability that $\mathcal{A}$ succeeds in foiling agreement (which is non-negligible). This contradicts the security of the signature scheme. ■

4.2 Sequentially Composable $\mathrm{ABG}_{n,n/2}$

Fitzi and Maurer [8] present a protocol for the Byzantine Generals problem that tolerates any $t < n/2$ faulty parties. Their protocol is for the information-theoretic and unauthenticated model. However, in addition to the point-to-point network, they assume that every triplet of parties is connected with an ideal (3-party) broadcast channel. As we have shown in Section 4.1, given a public-key infrastructure for signature schemes, it is possible to implement secure broadcast among three parties that composes sequentially. Thus, a protocol for $\mathrm{ABG}_{n,n/2}$ is derived by substituting the ideal 3-party broadcast primitive in the protocol of Fitzi and Maurer [8] with Protocol 3. Since Protocol 3 and the protocol of Fitzi and Maurer [8] both compose sequentially, the resulting protocol also composes sequentially.

THEOREM 3 *Assume that there exists a signature scheme that is existentially secure against chosen message attacks. Then, there exists a randomized protocol for authenticated Byzantine Generals that tolerates $t < n/2$ faulty parties and composes sequentially any polynomial number of times.*

As we show in Section 5, it is possible to execute many copies of an authenticated Byzantine Generals protocol concurrently, by allocating each execution a unique identifier that is common and known to all parties. Now, inside the Fitzi-Maurer protocol we can allocate unique indices to each invocation of the $\mathrm{ABG}_{3,1}$ protocol. We can therefore run the $\mathrm{ABG}_{3,1}$ protocols in parallel (rather than sequentially), improving the round complexity of the resulting protocol. In particular, our protocol is of the same round complexity as the underlying Fitzi-Maurer protocol. (We stress that the fact that the $\mathrm{ABG}_{3,1}$ subprotocols can be executed in parallel within the $\mathrm{ABG}_{n,n/2}$ protocol *does not* imply that the $\mathrm{ABG}_{n,n/2}$ protocol itself can compose in parallel. Rather, by our impossibility result, we know that it indeed *cannot* be composed in parallel.)

4.3 Sequentially Composable $\mathrm{ABG}_{n,t}$ for any t

In this section we describe a protocol for the Byzantine Generals problem for n parties, which can tolerate *any number of faulty parties*. However, this protocol is exponential in the number of participating parties. Therefore, in our setting, the protocol can only be carried out for $n = \log k$ parties (where k is a security parameter). We stress that the fact that the number of parties must be logarithmic in the security parameter is due to two reasons. First, we wish the protocol to run in polynomial time. Second, we use a signature scheme and this is only secure for polynomial-time adversaries, and a polynomial number of signatures.

Our protocol is constructed by presenting a transformation that takes a sequentially composable ABG protocol for $n-1$ parties which tolerates $n-3$ faulty parties, $\mathrm{ABG}_{n-1,n-3}$, and produces a sequentially composable ABG protocol for n parties which tolerates $n-2$ faulty parties, $\mathrm{ABG}_{n,n-2}$. Then, given our protocol for broadcast among three parties which tolerates one faulty party, $\mathrm{ABG}_{3,1}$, we can apply our transformation and obtain $\mathrm{ABG}_{n,n-2}$ for any n.

The idea for the transformation is closely related to the ideas behind the protocol for Byzantine Generals for three parties. The solution for the three-party broadcast assumes two-party broadcast (which is trivial). Using two-party broadcast, agreement on a fresh label can be reached. Having agreed on this label, the two point communications with the General are sufficient. Each party sends its claimed fresh label to the General, and the General includes the two received labels inside any signature that it produces. Our general transformation will work in the same manner. We use the $\mathrm{ABG}_{n-1,n-3}$ protocol to have all parties (apart from the General) agree on a random label. Then, each party privately sends this label to the General, who then includes all labels in its signatures. Thus, we prove:

THEOREM 4. *Assume that there exists a signature scheme that is existentially secure against chosen message attacks, for adversaries running in time* $\mathrm{poly}(k)$*. Then, there exists a Byzantine Generals protocols for $O(\log k)$ parties, that tolerates any number of faulty parties and composes sequentially.*

The formal description of the protocol and the proof of the theorem are omitted due to lack of space in this abstract.

5. AUTHENTICATED BYZANTINE AGREEMENT USING UNIQUE IDENTIFIERS

In this section we consider an augmentation to the authenticated model in which each execution is assigned a unique and common identifier. We show that in such a model, it is possible to achieve Byzantine Agreement/Generals that composes concurrently, for *any* number of faulty parties. We stress that in the authenticated model itself, it is not possible for the parties to agree on unique and common identifiers, without some external help. This is because agreeing on a common identifier amounts to solving the Byzantine Agreement problem, and we have proven that this cannot be achieved for $t \geq n/3$ when composition is required. Therefore, these identifiers must come from outside the system (and as such, assuming their existence is an augmentation to the authenticated model).

Intuitively, the existence of unique identifiers helps in the authenticated model for the following reason. Recall that our lower bound is based on the ability of the adversary to *borrow* signed messages from one execution to another. Now, if each signature also includes the session identifier, then the honest parties can easily distinguish between messages signed in this execution and messages signed in a different execution. It turns out that this is enough. That is, we give a transformation from almost any Byzantine Agreement protocol based on signature schemes, to a protocol that composes concurrently when unique identifiers exist. By "almost any protocol," we mean that this transformation applies for any protocol that uses the signature scheme for signing and verifying messages only. This is the natural use of the signature scheme and all known protocols indeed work in this way.

More formally, our transformation works as follows. Let Π be a protocol for authenticated Byzantine Agreement. We define a modified protocol $\Pi(id)$ that works as follows:

- Each party is given the identifier id as auxiliary input.
- If a party P_i has an instruction in Π to sign a given message m with its secret key sk_i, then P_i signs upon $id \circ m$ instead (where $\circ$ denotes concatenation).
- If a party P_i has an instruction in Π to verify a given signature σ on a message m with a public key pk_j, then P_i verifies that σ is a valid signature for the message $id \circ m$.

We now state our theorem:

THEOREM 5. *Let Π be a secure protocol for authenticated Byzantine Agreement which uses an existentially unforgeable signature scheme. Furthermore, this scheme is used for generating and verifying signatures only. Let the protocol $\Pi(id)$ be obtained from Π as described above, and let $id_1, \ldots, id_\ell$ be a series of ℓ unique strings. Then, the protocols $\Pi(id_1), \ldots, \Pi(id_\ell)$ all solve the Byzantine Agreement problem, even when run concurrently.*

We conclude by noting that it is not at all clear how such an augmentation to the authenticated model can be achieved in practice. In particular, requiring the on-line participation of a trusted party who assigns identifiers to every execution is clearly impractical. (Furthermore, such a party could just be used to directly implement broadcast.) However, we do note one important scenario where Theorem 5 can be applied. As we have mentioned, secure protocols often use many invocations of a broadcast primitive. Furthermore, in order to improve round efficiency, in any given round, many broadcasts may be simultaneously executed. The key point here is that *within* the secure protocol, unique identifiers can be allocated to each broadcast (by the protocol designer). Therefore, authenticated Byzantine Agreement can be used. Of course, this does not change the fact that the secure protocol itself will not compose in parallel or concurrently. However, it does mean that its security is guaranteed in the stand-alone setting, and a physical broadcast channel is not necessary.

6. OPEN PROBLEMS

Our work leaves open a number of natural questions. First, an unresolved question is whether or not it is possible to construct randomized protocols for authenticated Byzantine Generals that sequentially compose, for *any* n and any number of faulty parties. Second, it is unknown whether or not it is possible to construct a deterministic protocol that terminates in r rounds and sequentially composes ℓ times, for some $2 \leq \ell \leq 2r-1$. Another question that arises from this work is to find a realistic computational model for Byzantine Agreement that *does* allow parallel and concurrent composition for $n/3$ or more faulty parties.

Acknowledgments

We would like to thank Oded Goldreich for pointing out a simpler proof of Theorem 5. And Matthias Fitzi for discussions about [8].

7. REFERENCES

[1] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122, 1991.

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

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

[4] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-Box Concurrent Zero-Knowledge Requires Omega(log n) Rounds. In 33*th STOC*, pages 570–579. 2001.

[5] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *Crypto '00*, pages 74–92, 2000. LNCS No. 1880.

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

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

[8] M. Fitzi and U. Maurer. From partial consistency to global broadcast. In 32*th STOC*, pages 494–503. 2000.

[9] O. Goldreich. Concurrent Zero-Knowledge With Timing Revisited. In 34*th STOC*. 2002.

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

[11] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, Apr. 1988.

[12] L. Gong, P. Lincoln, and J. Rushby. Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults. In *Dependable Computing for Critical Applications*, 1995.

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

[14] S. Micali and P. Rogaway. Secure computation. In *Crypto '91*, pages 392–404, 1991. LNCS No. 576.

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

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

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

[18] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communication of the ACM, 21(2):120–126, 1978.