On the Composition of Authenticated Byzantine Agreement*

Yehuda Lindell† Anna Lysyanskaya‡ Tal Rabin§

January 3, 2007

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 (or Generals) and has been the focus of much research. Lamport, Shostack and Pease (1982) showed that in order to achieve Byzantine Agreement in the plain model, more than two thirds of the participating parties must be honest. They further showed that by augmenting the network with a public-key infrastructure for digital signatures, it is possible to obtain protocols that are secure for any number of corrupted parties. The problem in this augmented model is called "authenticated Byzantine Agreement".

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

1. Authenticated Byzantine Agreement protocols that remain secure under parallel or concurrent composition (even for just two executions) and tolerate a third or more corrupted parties, do *not* exist.
2. Deterministic authenticated Byzantine Agreement protocols that run for r rounds and tolerate a third or more corrupted parties, can remain secure for at most $2r-1$ sequential executions.

In contrast, we present randomized protocols for authenticated Byzantine Agreement that remain secure under sequential composition, for *any* polynomial number of executions. We exhibit two such protocols. In the first protocol an honest majority is required. In the second protocol, any number of parties may be corrupted; however, the complexity of the protocol is in the order of $2^n \cdot n!$ for n parties. In order to have this polynomial in the security parameter k (used for the signature scheme in the protocol), this requires the overall number of parties to be limited to $O(\log k / \log\log k)$. The above results are achieved due to a new protocol for authenticated Byzantine Generals for three parties that can tolerate any number of faulty parties and composes sequentially.

Finally, we show that when the model is further augmented so that in each session, all the participating parties receive a common session identifier that is unique to that session, then any polynomial number of authenticated Byzantine agreement protocols can be concurrently executed, while tolerating any number of corrupted parties.

Keywords: Authenticated Byzantine Agreement, protocol composition, lower bounds, randomized protocols.

---

*An extended abstract of this paper appeared in [29].

†Department of Computer Science, Bar-Ilan University, Ramat Gan, 52900, Israel. email: lindell@cs.biu.ac.il. Most of this work was carried out while the author was at the IBM T.J.Watson Research Center.

‡Computer Science Department, Brown University, Providence, RI 02912, USA. email: anna@cs.brown.edu. This work was carried out while the author was at the IBM T.J.Watson Research Center.

§IBM T.J.Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA. email: talr@watson.ibm.com.

1 Introduction

The Byzantine Agreement and Generals problems are amongst the most researched questions in distributed computing. Numerous variations of the problems have been considered under different communication models, and both positive results (i.e., protocols) and negative results (i.e., lower bounds on efficiency and fault tolerance) 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 primitive in its own right, broadcast is a key tool in the design of fault-tolerant distributed computing and secure protocols for multiparty computation [24, 4, 9].

The problem of Byzantine Generals is (informally) defined as follows. The setting is that of n parties connected via a point-to-point network, where one party is designated as the General (or dealer/sender) who holds an input message $x \in \{0,1\}$ that it wishes to broadcast to all the other parties. In addition, there is an adversary who controls up to t of the parties and can arbitrarily deviate from the designated protocol specification. The aim of the protocol is to securely simulate a broadcast channel. Thus, first and foremost, all (honest) parties must receive the *same* message. Furthermore, if the General is honest, then the message received by the honest parties must be x (i.e., the adversary is unable to cause honest parties to receive the wrong message when the General is honest). Finally, all honest parties are guaranteed to receive output (i.e., the adversary cannot carry out a "denial of service" attack). In the related Byzantine Agreement problem, all parties begin with an input and the aim is for all parties to "agree" on the same value, with a validity requirement that if all honest parties have the same input value, then this value must be the agreed-upon output. We note that in the case of an honest majority, these problems are essentially equivalent, in that a solution to one implies a solution to the other.

Pease et al. [31, 28] provided a solution to the Byzantine Agreement and Generals problems in the so-called plain model. In this model, a computationally unbounded adversary interacts with parties that are connected via point-to-point communication lines (and it is assumed that the adversary cannot block any messages that the parties send to each other). We remark that there is no setup phase assumed in this model. For their solution, the number of corrupted 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 that solves the Byzantine Agreement or Generals problem in the plain model can tolerate a third or more corrupted parties.

The above bound on the number of corrupted parties in the plain model is a severe limitation. It is therefore of great importance to find a different (and realistic) model in which it is possible to achieve higher fault tolerance. One possibility involves augmenting the plain model in a way that enables the authentication of messages that are sent by the parties. 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. We stress that what is needed here is the ability to prove to a third party the identity of the sender of a message. This can be achieved using a trusted preprocessing phase in which a public-key infrastructure for digital signatures (e.g. [34, 26]) is set up[1]. We call this the *authenticated* model. Pease et al. and Lamport et al. [31, 28], and later [13], use such an augmentation and obtain a protocol for the Byzantine Agreement and Generals problems that can tolerate *any* number of corrupted parties.

[1] We note that this typically requires that the adversary be computationally bounded. However, there exist preprocessing algorithms that do not require any computational assumptions; see [32].

1.1 Security under Composition

As stated, a common use of Byzantine Generals is to substitute a broadcast channel in multiparty protocols resulting in the BG protocol being executed many times. This is called protocol composition. Thus, we must consider the important question of whether the *authenticated* protocols also remain secure under such a composed execution. As we show in Section 2.3, it is not difficult to prove that the *unauthenticated* protocol of Pease et al. [31], and in fact all protocols in the plain model, do compose concurrently (and hence in parallel and sequentially). There are various issues that need to be addressed when discussing composition in general and specifically in the authenticated model.

1. COMMON SET-UP. As we mentioned, in order to enable signature capabilities there must be a trusted preprocessing phase to set it up. In the setting of composition we assume a common setup for all executions. We argue that this is the natural extension to the stand-alone authenticated model. An alternative extension is to have a different setup for each execution. However, these setups are costly in communication and computation and it is unrealistic to require trusted help for every protocol execution that takes place. We therefore consider a common preprocessing phase, meaning that all protocols use the same underlying mechanism and information in order to generate and verify signatures.

2. STATELESS AND UNIQUE IDS MODELS. In this work we consider two settings for the execution of a Byzantine Generals protocol. The first is a closed system with a central control entity that can issue IDs to processes. Every invocation of a protocol is considered as a sub-routine of the system itself and is provided with a unique ID. An example for this would be an auction system that is executed as stand alone protocol and invokes multiple BG protocols to which it transfers a unique ID for each invocation. The second setting is one with no central entity and thus protocols do not have distinguishing IDs. This latter case is called stateless because there is nothing that is common to different executions beyond the common setup. This is basically how protocols are executed in the Internet.

   We stress that in both settings, each protocol execution is run obliviously of the other executions taking place.

3. TYPES OF COMPOSITION. The variations in scheduling of the executions of the composed protocols yield different types of compositions: sequential, parallel and concurrent. In sequential composition, each execution begins strictly after the prior executions have terminated. In parallel composition, all executions begin at the same time and proceed at the same rate. Finally, in concurrent composition, the adversary has full control over when executions begin and at what rate they proceed. When only multiple invocations of a single protocol are being executed in the system this is also known as *self* composition.

Despite the importance of understanding the security of protocols under composition, most existing work (in both the standard and authenticated models) focused on the security and correctness of protocols in the stand-alone model only. The first to notice that composition of authenticated Byzantine Agreement and Generals with $t > n/3$ is problematic were Gong et al. [27], who also suggest methods for overcoming the problem. Our work shows that these suggestions and any others are in fact futile because composition in this model, without unique IDs and where more than a third of the parties are corrupt, is impossible.

1.2 Our Results

Stateless Authenticated Byzantine Generals. Our first result is a proof that stateless, authenticated Byzantine General/Agreement protocols (indeed even randomized) cannot be composed in parallel (and thus concurrently) using a common setup, unless more than two thirds of the parties are honest. Recall that authentication was added in order to circumvent the bound on the fault tolerance and did in fact achieve this in the stand-alone model. Yet, our result shows that despite popular belief, this enhancement *does not* provide the ability to improve fault tolerance when composition is required. That is, if there is a need for stateless parallel or concurrent composition, then the number of corrupted parties cannot be $n/3$ or more, and hence the authenticated model provides no advantage over the plain model. This result is summarized in the following theorem:

Theorem 1 *There does not exist a protocol for stateless authenticated Byzantine General/Agreement that remains secure under parallel composition (even for just two executions) when a third or more parties are corrupted.*

We prove Theorem 1 by showing that there cannot be a three-party protocol that composes in parallel even if two executions are being carried out. Then using the techniques of Lamport et al. [28] we extend this proof to give us the above theorem.

Theorem 1 relates to the infeasibility of parallel (and therefore concurrent) composition yet does not imply anything regarding sequential composition. Sequential composition is usually taken for granted, partly due to the fact that security under the standard (stand-alone) definitions for secure multiparty computation implies security under sequential composition [5]. However, this "sequential composition theorem" for secure computation only holds when no joint state between protocols is used. The theorem therefore does not necessarily hold for authenticated Byzantine Agreement, where the parties use the same public-key infrastructure in all executions. Note that, the existence of the public-key infrastructure is what enables higher fault tolerance in the stand-alone setting. However, it is also the source of difficulty under composition. Indeed, it turns out that the known protocols for authenticated Byzantine Agreement [31, 28, 13] do *not* remain secure under sequential composition, and can easily be "broken" in this setting.

We show that this is not a particular problem with the protocols of [31, 28, 13]. Rather, lower bounds actually hold for all *deterministic* protocols.[2] In particular, any deterministic protocol for authenticated Byzantine Agreement with a common setup that runs for r rounds and tolerates $t \geq n/3$ corrupted parties, can remain secure for at most $2r-1$ sequential executions. That is,

Theorem 2 *Let* Π *be a* deterministic *protocol for authenticated Byzantine Agreement that terminates within* r *rounds of communication and remains secure under sequential composition for* $2r$ *or more executions. Then* Π *can tolerate at most* $t < n/3$ *corrupted parties.*

In contrast, using *randomization* we obtain a protocol for authenticated Byzantine Generals with a common setup that remains secure for *any* polynomial number of sequential executions, and tolerates $t < n/2$ corrupted parties. Our protocol uses the (unauthenticated) Byzantine Agreement protocol of Fitzi and Maurer [17] who considered a model where the parties have access to an ideal three-party broadcast primitive. In this model, [17] present a Byzantine Generals protocol that tolerates $t < n/2$ corrupted parties.

[2] We note that our lower bound holds as long as the actual protocol is deterministic, even though the setup phase (say, for a public-key infrastructure for digital signatures) is randomized. Furthermore, when talking about signatures we will assume an ideal functionality of a signature operation (see [6] for such an ideal functionality) and prove our claims in that setting. Thus, we remove the randomness that can be associated with the signature.

In order to use the protocol of [17] we introduce a new authenticated Byzantine Generals protocol for three parties that remains secure under sequential composition. We prove the following theorem:

Theorem 3 *There exists a randomized protocol for authenticated Byzantine Generals for three parties that tolerates any number of faulty parties and remains secure under sequential composition.*

Substituting the protocol from Theorem 3 in the Fitzi-Maurer protocol (instead of the three-party broadcast primitive) yields an authenticated protocol that tolerates $t < n/2$ corrupted parties and remains secure under sequential composition. Thus, we prove:

Theorem 4 *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 $t < n/2$ corrupted parties and remains secure under sequential composition for any polynomial number of executions.*

We also present a randomized Byzantine Generals protocol that tolerates *any* number of corrupted parties, and remains secure under sequential composition for any polynomial number of executions. 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 (actually, the number of parties must be $O(\log k / \log \log k)$, where k is the security parameter). We note that this is not merely an issue of efficiency, but also of security. By definition, a signature scheme is only guaranteed to be secure when used a polynomial, in k, number of times, and when all parties including the adversary, run in polynomial-time. Despite the above, this protocol is interesting for two reasons. First, it suffices when the number of participating parties is "small". Second, it can be combined with the protocols of [10] in order to achieve the following result:

Theorem 5 *Assume that there exists a signature scheme that is existentially secure against chosen-message attacks. Then, for any constant $0 < \epsilon < 1$ there exists a randomized authenticated Byzantine Generals protocol for any number n parties, that tolerates $t < (1 - \epsilon)n$ corrupted parties and remains secure under sequential composition for any polynomial number of executions.*

Authenticated Byzantine Generals Using Unique Identifiers. As will be apparent from the proof of Theorem 1, the obstacle to achieving a secure solution in the setting of composition is the fact that honest parties cannot tell in which execution of the protocol a given message was sent, i.e. there is no "binding" between the message and a specific execution of the protocol. This allows the adversary to "borrow" messages from one execution and use them in another, enabling an attack on the protocols. However, if each protocol has a unique session identifier then this resolves the issue of the "binding" because the unique session ID can be signed together with the message. We thus prove:

Theorem 6 *Assume that there exists a signature scheme that is existentially secure against chosen-message attacks. Then, there exists a protocol for authenticated Byzantine Generals that tolerates any $t < n$ corrupted parties, and remains secure under concurrent composition in a model where unique session identifiers are allocated to every execution.*

The question of the existence of unique session identifiers is an interesting one, though out of the scope of this paper. However, as we have mentioned there are closed systems where such ID's are readily available. When talking about the Internet setting this seems to be a very strong assumption. For example, requiring the on-line participation of a trusted party who assigns identifiers to every execution is very difficult, if not impossible, to realize. Furthermore, if such a party existed it could just be used to directly implement the broadcast.

To conclude, our results strengthen the common belief that session identifiers are necessary for achieving authenticated Byzantine Generals. This is derived from the fact that when unique session identifiers are available they can be incorporated into the authenticated Byzantine Agreement to enable composition. Furthermore, our results demonstrate that if the system does not provide these identifiers by some centralized control, then such identifiers *cannot be generated by the parties in the system.* To see this, notice that by Theorem 6, if it were possible to obtain such session identifiers then it would be possible to achieve authenticated Byzantine agreement for any number of corrupted parties in the concurrent setting. However, this contradicts Theorem 1. Thus, one must assume the existence of some *trusted external means* for distributing unique and common identifiers.

1.3 Further Discussions

Stand-Alone versus Composition. Our result demonstrates that achieving security under composition (even a relatively weak type of composition such as two-execution parallel composition) can be strictly harder than achieving security in the stand-alone model. We note that such a "separation" between stand-alone security and security under composition can be made on many levels. One type of separation would state that protocols that are stand-alone secure are not necessarily secure under composition (this is the type of separation demonstrated regarding the parallel composition of zero-knowledge protocols [23] where in contrast [19] presents a parallel-secure protocol). A stronger separation may state that more *computational resources* are needed for achieving security under composition than in the stand-alone setting (the black-box lower bounds for concurrent zero-knowledge are actually separations of this kind; e.g., see [7] and in contrast [1]). However, the separation proven here is far stronger. It states that in the setting of composition, a certain *natural* problem (i.e., secure broadcast with a third or more corruptions, a common setup, and no common and unique identifiers) cannot be solved by *any* protocol, whereas solutions do exist for the stand-alone case. Thus, the additional difficulty of obtaining security under composition is not only with respect to protocol design, but also with respect to what problems can and cannot be solved.

Implications for Secure Multiparty Computation. In the setting of secure multiparty computation, a set of parties with possibly private inputs wish to carry out a computation on those inputs (classically, to compute a given function f of all of the inputs). The computation is required to preserve a number of security properties (privacy, correctness and more), even in the presence of malicious/Byzantine adversarial behavior. For more information on definitions and basic results, see [20, 21]. As we have mentioned, one important use for Byzantine Generals protocols is to substitute a broadcast channel that may be assumed in the design of multiparty protocols. In fact, many (if not most) protocols for secure multiparty computation assume a broadcast channel to simplify the description of the protocols (e.g. [24, 4, 9]). The implicit claim in these works was that this broadcast channel can be substituted by a Byzantine Generals protocol without any complications.

When the entire system is comprised of a single execution of a multiparty protocol, then the

substitution of the broadcast with a BG protocol is possible. This is trivially accomplished by having the MPC protocol locally generate unique IDs for its subprotocol invocations of BG. As the MPC protocol is the only protocol running in the system these IDs are unique for the whole system. However, when these multiparty protocols are executed concurrently in a system without a central control then the top level MPC protocols do not themselves have unique identifiers. Thus, when an MPC protocol calls the BG protocols as subroutines it can provide them with unique IDs within the specific MPC protocol, but clearly these IDs are not unique within the larger system in which the MPC protocol is running. Hence, Theorem 1 shows that the use of authenticated Byzantine Generals as a substitute for a broadcast channel in a top layer protocol prevents the composition of the top layer protocol, even if it is secure under composition when using a physical broadcast channel. Thus, in order to enable composition of multiparty protocols there either needs to be a physical broadcast channel or more than two thirds of the parties must be honest.

Related Work. The topic of protocol composition has received much attention. However, until recently, most of these works have focused on the problem of zero-knowledge and concurrent zero-knowledge, see [23, 12, 33, 7, 1] for just a few examples. It is instructive to compare our results to the work of Goldreich and Krawczyk [23] on zero-knowledge. They show that there exist protocols that are zero-knowledge when executed stand-alone, and yet do not remain zero-knowledge when composed in parallel (even twice). Thus, they show that zero-knowledge does not *necessarily* compose in parallel. However, zero-knowledge protocols that compose in parallel do exist (for example, see [19]). In contrast, we show that it is impossible to obtain *any protocol* for Byzantine Agreement that will compose in parallel (even twice).

If we allow various relaxations to the model then the impossibility results of this work can be circumvented. Motivated by our work, [2] show that if you are willing to remove the requirement that honest parties must always successfully complete the protocol and receive output, then it is possible to generate unique identifiers in the concurrent setting. Goldwasser and Lindell [25] show how a minor relaxation of the requirements on secure computation can be used in order to replace a broadcast channel by a simple two-round echo-broadcast protocol that remains secure under composition. In similar work [15, 16], it was shown how to construct (randomized) protocols for *weak* Byzantine Generals that tolerate any number of corrupted parties and do not require any setup assumptions. Furthermore, it was shown how to use these protocols to obtain secure computation (without the relaxation required by [25]).

For other work on the topic of protocol composition, mainly related to general secure multiparty computation, we refer the reader to [3, 30, 5, 11, 18, 6]. We also remark on an interesting connection between our work and the UC composition theorems of [6, 8]. In the UC framework of [6], it is assumed that there exist unique and common session identifiers for every execution, and this is utilized in an essential way for proving the security of protocols. To some extent, our work can be seen as an explanation as to why unique session identifiers are needed in this framework.

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. Each party is formally modeled by an interactive Turing machine with $n-1$ pairs of incoming and outgoing communication tapes (one

pair for each party). 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 outgoing communication tapes, and in the *receive* phase, the parties read the contents of their incoming communication 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 a signature scheme to authenticate messages that they receive (see Section 2.4), and is thus the source of the name "authenticated". However, we stress that our lower bound holds for all possible preprocessing processes, even those that cannot be efficiently generated. See Definition 3 for a formal definition of this model.

The adversarial model that we consider is one where the adversary controls up to $t \leq n$ of the parties $P_1, \ldots, P_n$ for some threshold parameter t. These adversary-controlled parties are said to be corrupted. The adversary receives the corrupted parties' views and determines the messages that they send. These messages need not be according to the protocol execution, but rather can be computed by the adversary as an arbitrary (efficiently computable) function of its view.

Our protocols for authenticated Byzantine Agreement that remain secure under sequential composition rely on the security of signature schemes, and thus assume that adversaries (and, of course, honest parties) are limited to probabilistic polynomial-time. In contrast, 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. Therefore, our lower bounds hold also for (inefficient) protocols where the honest parties do an exponential amount of work.

In this paper we consider two different models of corruption: adaptive corruptions and static (non-adaptive) corruptions. In the adaptive adversarial model, the adversary can choose to corrupt parties at any point during the computation, with the only limitation being that at most t parties may be corrupted. Furthermore, the choice of which parties to corrupt (if any) can be made adaptively, as a function of its view. In contrast, in the static adversarial model, the set of at most t corrupted parties is fixed before the computation begins. We prove our protocols for adaptive adversaries and our lower bounds for static adversaries (thereby strengthening the results).

2.2 Byzantine Generals and Byzantine Agreement

The existing literature defines two related problems: Byzantine Generals and Byzantine Agreement. In the Byzantine Generals problem, there is one designated party, the General or dealer, who wishes to broadcast its value to all the other parties. The requirements on such protocols are *agreement* (meaning that all honest parties must output the same value) and *validity* (meaning that if the dealer is honest, then all honest parties output its value correctly). Thus, stated differently, the Byzantine Generals problem is that of achieving secure broadcast. In the Byzantine Agreement problem, each party has an input and the parties wish to agree on a value. Here too the requirements are *agreement* (meaning the same as above) and *validity*. However, in the Byzantine Agreement problem, the validity condition states that if a majority of the parties are honest and they all begin with the same value, then they must terminate with that value.

Byzantine Generals versus Byzantine Agreement. The Byzantine Generals and Agreement problems are closely related. First, any protocol for Byzantine Generals (or secure broadcast) can be used to solve the Byzantine Agreement problem. Furthermore, when assuming an honest majority (in which case the Byzantine Agreement validity condition applies), the other direction also holds. However, in the general case, where any number of parties may be corrupted, it is not known whether or not a Byzantine Agreement protocol can be used to solve the Byzantine Generals problem. This disparity is due to the fact that the *validity condition* of the Byzantine Agreement problem only has meaning when a majority of the parties are honest, whereas the validity requirement for the Byzantine Generals problem is also applicable when a majority of the parties are corrupted. Therefore, when there is no honest majority these two notions cannot be compared as Byzantine Agreement is undefined while the Byzantine Generals is.

Given the above relationships between BG and BA in this paper, we prove our impossibility results and protocols for the Byzantine Generals problem. This provides the most generality as our protocols yield protocols also for Byzantine Agreement. Furthermore, our impossibility results for Byzantine Generals also apply to Byzantine Agreement for the case of an honest majority.

In the definitions below, 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 as there could be randomized protocols that would solve the problem, but might introduce a probability of error. Formally, the Byzantine Generals problem is defined as follows:

Definition 1 (Byzantine Generals – stand-alone): *Let $P_1, \ldots, P_{n-1}$ and $G = P_n$ be n parties, let G be a designated party with input $x \in \{0,1\}$ and let $\mathcal{A}$ be an (adaptive or static) adversary who can control up to t of the parties, including G. Then, a protocol solves the* Byzantine Generals *problem, tolerating t corruptions, if for any adversary $\mathcal{A}$ the following two properties hold (except with negligible probability):*

1. Agreement: *All honest parties output the same value.*
2. Validity: *If G is honest and has input x, then all honest parties output x.*

We denote a protocol for n parties that tolerates t corruptions by $\mathsf{BG}_{n,t}$.

In the setting of Byzantine Agreement, the validity property states that if "enough" of the parties are honest (i.e., uncorrupted), and they all begin with the same input value, then they must output that value.

Definition 2 (Byzantine Agreement – stand-alone): *Let $P_1, \ldots, P_n$ be n parties with associated inputs $x_1, \ldots, x_n$ and let $\mathcal{A}$ be an (adaptive or static) adversary who can control up to t of the parties. Then, a protocol solves the* Byzantine Agreement *problem, tolerating t corruptions, if for any adversary $\mathcal{A}$ the following two properties hold (except with negligible probability):*

1. Agreement: *All honest parties output the same value.*
2. Validity: *If more than $n/2$ parties are honest, and they all have the same input value x, then all honest parties output x.*

Authenticated Byzantine protocols: In the model for authenticated Byzantine Generals and 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 – stand-alone): *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 the protocol execution, a 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. The requirements of Agreement and Validity need to be satisfied as in the non-authenticated setting. These properties are achieved with high probability of error and in this case the probability of error is taken also over the choices of the set-up tape. We denote an authenticated Byzantine Generals protocol for n parties that tolerates t corruptions by $\mathsf{ABG}_{n,t}$.

It is crucial to note that the different s_i values in the setup phase may be dependent. Therefore, given s_i it is not (necessarily) possible to sample the s_j values for $j \neq i$. This is a crucial point that illustrates why the proof of Proposition 2.1 below (that relates to the composability in the plain unauthenticated setting) does not carry over to the authenticated model. Of course, if it were possible to sample the s_j values oneself, there would be no need for an external, trusted phase. Rather, the parties could carry out the setup themselves. 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 for digital signatures. That is, the trusted party chooses verification and signing key-pairs $(vk_1, sk_1), \ldots, (vk_n, sk_n)$ from a secure signature scheme, and sets the contents of party P_i's tape to equal $s_i(sk_i, vk_1, \ldots, \ldots, vk_n)$. In other words, all parties are given their own signing key and the verification keys of all the other parties (the dependence mentioned above is very evident in this classic case). We note that this preprocessing phase can also be used to choose a *common reference string* to be accessed by all parties (in this case, all the s_i's are set to the same reference string).

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. On the other hand, our positive results hold with an efficiently implemented set-up.

2.3 Composition of Protocols

This paper deals with the security of authenticated Byzantine Generals/Agreement protocols under composition. In general, the notion of "protocol composition" refers to a setting where the participating parties are involved in many protocol executions. Furthermore, the honest parties participate in each execution as if it is running in isolation (and therefore obliviously of the other executions taking place). In particular, this means that honest parties are not required to coordinate between different executions or remember the history of past executions. Requiring parties to coordinate between executions is highly undesirable and sometimes may even be impossible (e.g., it is hard to imagine successful coordination between protocols that are designed independently

of each other). In contrast to the honest parties, we assume that the adversary may coordinate its actions between the protocol executions. This asymmetry is due to the fact that some level of coordination is clearly possible. Thus, although it is undesirable to rely on it in the construction of protocols, it would be careless to assume that the adversary cannot utilize it to some extent. This notion of protocol composition is called *stateless composition*. We note that in stateless composition, the parties run identical copies of the protocol in each execution. This implies, for example, that there is no information that is externally provided to the parties and is unique for every execution, like a common session identifier. (See the Introduction for a discussion on session identifiers and their role.) Formally, composition is captured by the following process:

Definition 4 (Composition): *Let $P_1, \ldots, P_n$ be parties for an authenticated Byzantine Generals/Agreement protocol (ABG/ABA) Π. Then,* Π remains secure under sequential (resp., parallel or concurrent) composition *if for every polynomial-time (adaptive or static) adversary $\mathcal{A}$, the requirements for Byzantine Generals/Agreement hold for Π for every execution within the following process:*

- *Run the preprocessing phase associated with Π and obtain the strings $s_1, \ldots, s_n$. Then, for every i, set the* setup-tape *of P_i to equal s_i.*
- *Repeat the following process sequentially (resp., in parallel or concurrently) until the adversary halts:*
  1. *The adversary $\mathcal{A}$ chooses:*
     - *ABG - input x for party (w.l.o.g) $P_1 = G$*
     - *ABA - inputs $x_1, \ldots, x_n$ for parties $P_1, \ldots, P_n$.*
  2. *For every honest party P_i, the value x_i is written on its input tape, and a uniformly (and independently) chosen random string is written on its random-tape.*
  3. *All parties are invoked for an execution of Π (using the strings generated in the preprocessing phase above). As in a stand-alone execution, the messages sent by the corrupted parties are determined by the adversary $\mathcal{A}$, whereas all other parties follow the instructions of Π.*

We stress that the preprocessing phase is executed only once, and that all executions use the strings distributed in this phase.[3] This fact means that there is a dependency between the executions, which is a crucial point in the proof of the lower bounds. If we were to have a separate and independent preprocessing phase for each invocation of the protocol then the lower bounds would no longer hold (of course, such an assumption is highly unreasonable in most settings). 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 setup tapes (i.e., the composition is *stateless*). In contrast, the adversary $\mathcal{A}$ *can* coordinate between the executions, and its view at any given time includes all the messages received in all the executions. Finally, notice that the adversary is given the power to adaptively choose the inputs in each execution. We remark that our impossibility results hold even if all the inputs are fixed ahead of time.

[3] 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.

Before continuing, we show that any Byzantine Generals (or Agreement) protocol *in the standard model*, (i.e. unauthenticated) composes concurrently. Intuitively, this holds because the parties have no secret input and so all but one execution can be internally simulated, thereby reducing the setting of composition to the stand-alone setting[4]. In contrast, our work focuses on the difficulties caused by composing *Authenticated* Byzantine Generals (or Agreement).

Proposition 2.1 *Let Π be a protocol that solves the Byzantine Generals (resp. Agreement) problem in the* plain model *and tolerates t corruptions. Then, Π solves the Byzantine Generals (resp., Agreement) problem under concurrent composition, and tolerates t corruptions.*

Proof (sketch): 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 do not meet the requirement on a Byzantine Generals (resp., Agreement) protocol. Then, 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 uniformly at random an index $i \in \{1, \ldots, N\}$. 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 $\mathcal{A}'$). However, for the i^{th} execution, $\mathcal{A}'$ externally interacts with the honest parties and passes messages between them and $\mathcal{A}$. The key point to notice is that the honest parties hold no secret inputs and run each execution independently of the others. 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 with success probability that equals $1/N$ times the success probability of $\mathcal{A}$. This contradicts the stand-alone security of Π. ■

Extensions. Notice that the proof of Proposition 2.1 goes through as long as the adversary $\mathcal{A}'$ is able to simulate the actions of the honest parties in all executions, except one. Thus, it is possible to extend Proposition 2.1 to the setting where an *independent setup* is provided for each execution, and so the adversary $\mathcal{A}'$ can still simulate all of the executions except for the i^{th} one. This demonstrates that our impossibility results are really due to the use of a common setup for all executions. (We note that this extension is of theoretical interest only; it is hard to conceive of any setting where an independent setup can be generated for every execution.)

2.4 Signature Schemes

Our protocols for authenticated Byzantine Agreement use secure signature schemes. Formally[5], a signature scheme is a triplet of algorithms (Gen, S, V), where Gen is a probabilistic generator that outputs a pair of verification and signing keys (vk, sk), S is a signing algorithm and V is a verification algorithm. The validity requirement for signature schemes states that except with negligible probability, for every message m, $V(vk, m, S(sk, m)) = 1$, where $(vk, sk) \leftarrow Gen(1^n)$; i.e., honestly generated signatures are almost always accepted. We sometimes denote $V(vk, \cdot)$ by $V_{vk}(\cdot)$ and likewise for S.

The security requirement of a signature scheme states that the probability that an efficient forging algorithm $\mathcal{A}$ can succeed in generating a valid forgery is negligible. This should hold even

[4] Notice that there is no privacy requirement whatsoever for these problems; therefore, even the parties' inputs are not secret. Indeed, by our formulation, the adversary explicitly chooses them.

[5] We follow standard formulations such as [26].

when $\mathcal{A}$ is given oracle access to a signing oracle (this oracle represents valid signatures that $\mathcal{A}$ may obtain). In order for $\mathcal{A}$ to succeed, it must generate a valid signature on a message that was not queried to the signing oracle. More formally, the following experiment is defined: The generator Gen is run, outputting a key-pair (vk, sk). Then, $\mathcal{A}$ is given vk and oracle access to the signing oracle $S(sk, \cdot)$. At the conclusion of the experiment, $\mathcal{A}$ outputs a pair (m^*, σ^*). Let Q_m be the set of oracle queries made by $\mathcal{A}$. Then, we say that $\mathcal{A}$ has succeeded, denoted $\mathsf{succeed}_{\mathcal{A}}(vk, sk) = 1$, if $V(vk, m^*, \sigma^*) = 1$ and $m^* \notin Q_m$. That is, $\mathcal{A}$ outputs a message along with a valid signature, and $\mathcal{A}$ did not query its oracle with this message. We are now ready to present the formal definition of security for signature schemes:

Definition 5 *A signature scheme (Gen, S, V) is* existentially secure against chosen-message attacks *if for every probabilistic polynomial-time adversary $\mathcal{A}$, every polynomial $p(\cdot)$ and all sufficiently large k's*

$$\Pr_{(vk,sk)\leftarrow Gen(1^k)}[\mathsf{succeed}_{\mathcal{A}}(vk, sk) = 1] < \frac{1}{p(k)}$$

3 Impossibility for Parallel Composition

In this section we show that it is impossible to construct an authenticated Byzantine Generals protocol that remains secure under parallel or concurrent composition, and tolerates a third or more of the parties being corrupted. This result is analogous to the lower bounds for Byzantine Generals and Agreement in the plain model, without authentication [31, 28]. We stress that our result does not merely show that authenticated Byzantine Generals protocols do not necessarily remains secure under composition; rather, we show that one *cannot* construct protocols that will be secure in this setting. The results below are stated for static adversaries and therefore also apply to adaptive adversaries.

Intuition. Let us first provide some intuition into why the added power of the preprocessing step in authenticated Byzantine Generals does not help when composition is required. Recall that this added power is the reason that Authenticated Byzantine Generals can tolerate more faults in the standard stand-alone setting. An instructive step is to first see how authenticated Byzantine Generals protocols typically utilize authentication (i.e., digital signatures) in order to increase fault tolerance. Consider three parties G, A and B participating in a standard (unauthenticated) Byzantine Generals protocol. Furthermore, assume that during the execution A claims to B that G sent it some message x. Then, B cannot differentiate between the case that G actually sent x to A, and the case that G did not send this value and A is lying. Thus, B cannot be sure that A really received x from G. Indeed, the plain model has been called the "oral message" model, in contrast to the "signed message" model of authenticated Byzantine Agreement [28]. The use of signature schemes helps to overcome this exact problem: If G had *signed* the message x and sent this signature to A, then A could forward the signature to B. Since A cannot forge G's signature, this would then constitute a proof that G had indeed sent x to A. Utilizing the unforgeability property of signatures, it is thus possible to achieve Byzantine Generals for any number of corrupted 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 G's signature on x, yet B would still not know whether G signed x in this execution, or in a different (concurrent or parallel) execution. Thus, the mere fact that A produces G's signature on a value does not provide proof that G signed *this* value in *this* execution. As we

will see in the proof, this is enough to render the public-key infrastructure (and in general, any set-up) "useless" under parallel composition.

We start by proving the following lemma:

Lemma 3.1 *There does not exist a protocol for authenticated Byzantine Generals with three parties that remains secure under composition of two parallel executions while tolerating a single statically corrupted party.*

Using the techniques of Lamport et al. [28] we extend the above lemma to show that there does not exist a protocol for ABG with n parties with more than a third of the parties being corrupted. Furthermore, as our lemma states the most relaxed requirement for the composition, i.e. parallel composition of only two executions, we get the strongest result possible, that is, that no composition is possible. Thus we get the following theorem:

Theorem 1 *Restated. There does not exist a protocol for authenticated Byzantine Generals with n parties that remains secure under parallel (and thus concurrent) composition which can tolerate $n/3$ or more statically corrupted parties.*

Proof: Our proof of Lemma 3.1 uses ideas from the proof by Fischer et al. [14] that no unauthenticated Byzantine Agreement protocol with three parties can tolerate one or more corrupted parties.

Assume, by contradiction, that there exists a protocol Π that solves the Byzantine Generals problem for three parties G, A and B, where one may be corrupt. Furthermore, Π remains secure even when composed in parallel twice. Exactly as in the proof of Fischer et al. [14], we define an imaginary hexagonal system S that "intertwines" two independent executions of Π. That is, let G_0, A_0, B_0 and G_1, A_1 and B_1 be independent copies of the three parties participating in Π. By independent copies, we mean that G_0 and G_1 are the same party G with the same setup-tape, that runs in two different parallel executions of Π, as defined in Definition 4. The system S is defined by connecting party A_0 to B_1 (rather than to B_0) and party B_0 to A_1 (rather than to A_0), as detailed in Figure 1.

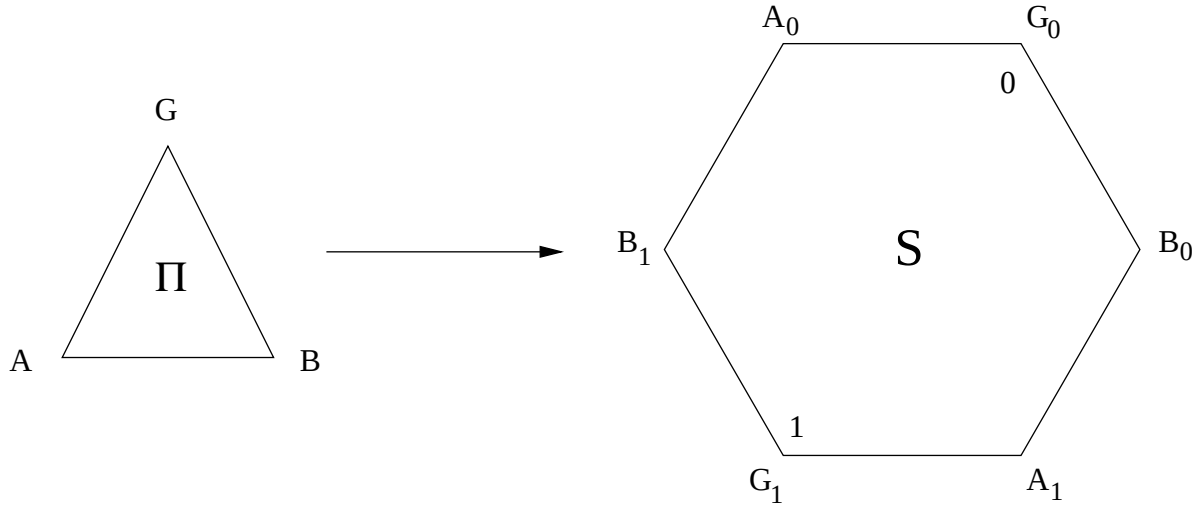

Figure 1: Combining two copies of Π in a hexagonal system S.

In the system S, party G_0 has input 0 and party G_1 has input 1, and within S, all parties follow the instructions of Π exactly. We stress that S is *not* a Byzantine Generals setting (where the

parties are joined in a complete graph on three nodes), and therefore the definitions of Byzantine Generals 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 Generals protocol, then we arrive at a contradiction regarding the output distribution in S. We begin by showing that G_0 and A_0 always output 0 in S. (Later we will show other equalities and the contradiction will follow.)

We denote by $\mathsf{rounds}(\Pi)$ the upper bound on the number of rounds of Π (when run in a Byzantine Generals setting). By the termination requirement on Byzantine Generals protocols, $\mathsf{rounds}(\Pi)$ is well-defined (and finite). We note that by the contradicting assumption, Π is secure under parallel composition, and so must halt within $\mathsf{rounds}(\Pi)$ rounds when run in parallel.

Claim 3.2 *Except with negligible probability, parties G_0 and A_0 halt within* $\mathsf{rounds}(\Pi)$ *steps and output* 0 *in the system S.*

Proof:

The intuition behind the proof is very simple. We need to show that utilizing two parallel executions of Π we can generate the system S, and then borrowing from the requirements of the ABG's we deduce what output the parties in S should have. That is we show that the view of G_0 and A_0 in S is identical to that of two honest parties in an ABG protocol, where the honest general's input is 0, thus based on the ABG's requirements means that the output of the two parties needs to be 0. Ti is visually easy to see this claim. Examine the hexagon S and virtually collapse both B_0 and B_1 into one party, this shows us how the messages in S are taken from the two parallel executions. The faulty party's only malicious actions are redirection of messages, rather than actual faulty behavior. Thus, the following formal proof labors to show first that the honest parties, G_0 and A_0, are in fact sending messages that they would send as honest participants in an authenticated Byzantine Generals protocol and second that the exact messages which the faulty party sends are dictated by the system S. This is achieved by showing that a simple redirection of messages between two parallel executions of an authenticated BG suffices to emulate the system S.

We show that there exists an adversary $\mathcal{B}$ who controls the two parties B_0 and B_1 who participates in two parallel copies of Π and emulates the system S, with respect to G_0 and A_0's view. The adversary $\mathcal{B}$ (and the other honest parties participating in the parallel executions) work within a Byzantine Generals 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.

Before formally proving the above, we introduce some terminology and notation. A system X is defined by a set of parties along with protocol instructions for these parties, an adversary along with a set of corrupted parties, and a set of inputs for all parties. In addition, part of the system definition includes the network structure by which the parties are connected. Let X be a system and let P be a party in X. Then, $\mathsf{view}_X(P)$ denotes the view of party P in an execution of X; this view contains the contents of P's input tape and random tape, along with the series of messages that it received during the execution. If the parties within the system are deterministic (including the adversary), then $\mathsf{view}_X(P)$ is a single value (since all messages are predetermined by the parties' strategies and their inputs). If the parties are probabilistic, then $\mathsf{view}_X(P)$ is a random variable assuming values over P's view, when all parties' random tapes are chosen uniformly at random.

We define two different systems and prove some properties of these systems. First, let S be the above-defined hexagonal system involving parties G_0, G_1, A_0, A_1, B_0, and B_1. As described G_0 has input 0 and G_1 has input 1. The parties instructions are to honestly follow the protocol instructions of Π, that is each party generates and sends messages as it would honestly do in an execution of

an authenticated Byzantine Generals protocol (A_i behaves like A, etc.). The network structure is as shown in Figure 1. (In this system, there is no adversary; formally, the set of corrupted inputs is empty.)

Next, we define a system 2BA that is made up of two parallel executions of Π (thus defining the network structure); denote these two executions by Π_0 and Π_1. In the first execution (i.e., in Π_0), party G_0 has input 0 and interacts with (i.e. is connected to) A_0 and B_0; in the second execution (i.e., in Π_1), party G_1 has input 1 and interacts with (i.e. is connected to) A_1 and B_1. Recall that B_0 and B_1 are independent copies of the party B with the same key tape (as in Definition 4); likewise for G_0, G_1 and A_0, A_1. The uncorrupted parties in the system 2BA honestly follow the instructions of Π. Now, we consider the adversary in these two executions. Let $\mathcal{B}$ be the adversary and let the set of corrupted parties be B_0 and B_1. Intuitively, party $\mathcal{B}$'s strategy is to maliciously generate an execution in which G_0's and A_0's view in 2BA is *identical* to their view in S. That is, we will construct $\mathcal{B}$ such that $\mathsf{view}_{\mathsf{2BA}}(G_0)$ is distributed exactly according to $\mathsf{view}_S(G_0)$; likewise, $\mathsf{view}_{\mathsf{2BA}}(A_0)$ will have the same distribution as $\mathsf{view}_S(A_0)$. (We remark that if Π is deterministic, then we will have that $\mathsf{view}_{\mathsf{2BA}}(G_0) = \mathsf{view}_S(G_0)$ and $\mathsf{view}_{\mathsf{2BA}}(A_0) = \mathsf{view}_S(A_0)$.) We now describe how $\mathcal{B}$ achieves this: $\mathcal{B}$ works by redirecting edges in the two parallel triangles (representing two parallel executions), so that the overall system has the same behavior as S; see Figure 2.

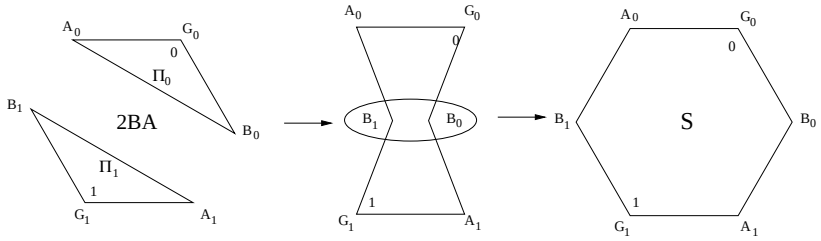

Figure 2: Collapsing B_0 and B_1 into a single faulty entity.

Specifically, in 2BA the (A_0, B_0) and (A_1, B_1) edges of Π_0 and Π_1 respectively are effectively removed, and the (A_0, B_1) and (A_1, B_0) edges of S are effectively added in their place. $\mathcal{B}$ is able to make such a modification because it only involves redirecting messages to and from parties that it controls (i.e., B_0 and B_1). Recall that the corrupted party can coordinate between the different executions. We now formally describe how $\mathcal{B}$ works (in the description below, $\mathsf{msg\text{-}2BA}_i(P_j, P_\ell)$ denotes the message sent from party P_j to party P_ℓ in the i^{th} round of the 2BA execution):

$\mathcal{B}$ invokes parties B_0 and B_1. We stress that B_0 and B_1 follow the instructions of protocol Π exactly. However, $\mathcal{B}$ provides them with their incoming messages and sends their outgoing messages for them. The only malicious behavior of $\mathcal{B}$ is in the redirection of messages to and from B_0 and B_1. A full description of $\mathcal{B}$'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:* $\mathcal{B}$ obtains messages $\mathsf{msg\text{-}2BA}_i(B_0, G_0)$ and $\mathsf{msg\text{-}2BA}_i(B_0, A_0)$ from B_0 in execution Π_0, and messages $\mathsf{msg\text{-}2BA}_i(B_1, G_1)$ and $\mathsf{msg\text{-}2BA}_i(B_1, A_1)$ from B_1 in Π_1 (these are the round i messages sent by B_0 and B_1 to the other parties in system 2BA; as we have mentioned, B_0 and B_1 compute these messages according to the protocol definition and based on their view).

   - In Π_0, $\mathcal{B}$ sends G_0 the message $\mathsf{msg\text{-}2BA}_i(B_0, G_0)$ (this message was indeed intended for G_0), and sends A_0 the message $\mathsf{msg\text{-}2BA}_i(B_1, A_1)$; notice that this message was actually intended for A_1.

- In Π_1, $\mathcal{B}$ sends G_1 the message $\mathsf{msg\text{-}2BA}_i(B_1, G_1)$ and sends A_1 the message $\mathsf{msg\text{-}2BA}_i(B_0, A_0)$.

2. *Obtain incoming messages from round i:* $\mathcal{B}$ receives messages $\mathsf{msg\text{-}2BA}_i(G_0, B_0)$ and $\mathsf{msg\text{-}2BA}_i(A_0, B_0)$ from G_0 and B_0 in round i of Π_0, and messages $\mathsf{msg\text{-}2BA}_i(G_1, B_1)$ and $\mathsf{msg\text{-}2BA}_i(A_1, B_1)$ from G_1 and A_1 in round i of Π_1.

   - $\mathcal{B}$ passes B_0 in Π_0 the message $\mathsf{msg\text{-}2BA}_i(G_0, B_0)$, i.e. the message that was originally intended for it, and the message $\mathsf{msg\text{-}2BA}_i(A_1, B_1)$ that was intended for B_1.
   - $\mathcal{B}$ passes B_1 in Π_1 the messages $\mathsf{msg\text{-}2BA}_i(G_1, B_1)$ and $\mathsf{msg\text{-}2BA}_i(A_0, B_0)$.

This completes the description of $\mathcal{B}$, and thus the definition of the system 2BA.

We now claim that $\mathsf{view}_{\mathsf{2BA}}(G_i)$, $\mathsf{view}_{\mathsf{2BA}}(A_i)$ and $\mathsf{view}_{\mathsf{2BA}}(B_i)$ are distributed exactly like $\mathsf{view}_S(G_i)$, $\mathsf{view}_S(A_i)$ and $\mathsf{view}_S(B_i)$. We prove the claim by induction on the number of steps. Step 1. As the parties locally follow the protocol correctly all parties in both 2BA and S generate the same set of messages to be sent out. In S the messages are sent according to the connectivity of the hexagon. In 2BA the adversary redirects to B_1 the messages intended to be sent by A_0 to B_0; to A_0 the messages intended to be sent by B_1 to A_1; to B_0 the messages intended to be sent by A_1 to B_1; to A_1 the messages intended to be sent by B_0 to A_0. An examination of the messages received by each party in S and 2BA shows that the distributions of the views of all parties at the end of this step in the two executions are identical. After j steps the distribution of the views of the parties in the two systems is identical. In step $j+1$ the parties first generate the messages that they intend to send out. The distribution of their views in both systems is the same they generate messages with the same distribution, as they are correctly following the protocol. Again, as in the case of step 1, the messages will be sent in S according to the hexagon, which exactly matches the redirection of messages in 2BA.[6]

By the assumption that Π is a correct Byzantine General protocol that remains secure under parallel composition (even if for only two executions), we have that in execution Π_0 of 2BA, both G_0 and A_0 halt within $\mathsf{rounds}(\Pi)$ steps and output 0 (except with negligible probability). The fact that they both output 0 is derived from the fact that G_0 is honest and has input 0. Therefore, they must output 0 in the face of *any* adversary $\mathcal{B}$ controlling B_0; in particular this holds with respect to the specific adversary $\mathcal{B}$ described above. (We stress that it is very possible that G_0 and A_0 detect adversarial behavior. However, the security requirements for Byzantine Generals are such that even in such a case, the parties must halt with "correct" output.) Since the views of G_0 and A_0 in S are identically distributed to their views in Π_0, we conclude that in the system S they also halt within $\mathsf{rounds}(\Pi)$ steps and output 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 G_1 and B_1 halt within* $\mathsf{rounds}(\Pi)$ *steps and output* 1 *in the system S.*

In order to prove this claim, the adversary is $\mathcal{A}$ who controls A_0 and A_1 and works in a similar way to $\mathcal{B}$ in the proof of Claim 3.2 above. (The only difference is regarding the edges that are redirected.)

[6] We note a crucial difference between this proof and that of Fischer et al. [14]. In [14], the corrupted party $\mathcal{B}$ is able to simulate the entire B_0–G_0–A_0–B_1 segment of the hexagon system S by itself. Thus, in a *single* execution of Π with G_0 and A_0, party $\mathcal{B}$ can simulate the hexagon. Here, due to the fact that the parties G_0 and A_0 may have secret information that $\mathcal{B}$ does not have access to, $\mathcal{B}$ is unable to simulate their behavior itself. Rather, $\mathcal{B}$ needs to redirect messages from the parallel execution of Π_1 in order to complete the hexagon.

Claim 3.4 *Except with negligible probability, parties A_0 and B_1 halt within* rounds(Π) *steps and output the same value in the system S.*

Similarly, this claim is proven by defining an adversary $\mathcal{G}$ who controls G_0 and G_1 and follows a similar strategy to $\mathcal{B}$ 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 A_0 must output 0 in S (Claim 3.2), and B_1 must output 1 in S (Claim 3.3). On the other hand, by Claim 3.4, parties A_0 and B_1 must output the same value. We conclude that there does not exist a 3-party protocol for Byzantine Generals that tolerates one corruption and composes twice in parallel. This concludes the proof of the lemma. ■

4 Sequential Composition of Deterministic Protocols

Theorem 1 states that it is impossible to obtain an authenticated Byzantine Generals protocol that remains secure under parallel composition if more than a third of the parties are faulty. The impossibility result holds even if only two parallel executions take place and even for randomized protocols. In this section, we consider a much more benign type of composition and a limited class of protocols. That is, we study the possibility of obtaining *deterministic* protocols for authenticated Byzantine Generals that remain secure under *sequential* composition. We note that the protocol itself must be deterministic, but the setup phase is allowed to be randomized. For more details see Footnote 2.

We show that any such protocol that terminates within r rounds of communication can remain secure for (at most) $2r-1$ sequential executions. Thus, efficient deterministic protocols that remain secure for any polynomial number of executions *do not exist.* Furthermore, since the number of rounds in the protocol must be at least linear in the number of times it is to compose, we rule out the possibility of obtaining deterministic protocols that use few rounds and are secure for a large number of sequential executions. We remark that the currently known protocols for (stand-alone) authenticated Byzantine Agreement are deterministic [31, 28]. Furthermore, these protocols can actually be completely broken in the second execution.

The lower bound here 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 Generals protocol Π. However, a contradiction is only derived if the system S halts. Since Π terminates within r rounds, the system S also halts within r rounds. Therefore, a contradiction is reached if Π is run $2r$ or more times. We conclude that the protocol Π can remain secure for at most $2r-1$ sequential executions.

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 *(Restated.) Let* Π *be a* deterministic *protocol for authenticated Byzantine Generals that terminates within r rounds of communication and remains secure under sequential composition for $2r$ or more executions. Then* Π *can tolerate at most $t < n/3$ statically corrupted parties.*

Proof: As we have mentioned, we prove this theorem by showing that a corrupted party in a (sequential execution of a) deterministic authenticated Byzantine Generals protocol Π for three parties, can perfectly simulate r rounds of the hexagonal system S, using $2r$ sequential executions. Thus the proof here is very similar to the proof of Theorem 1. The main difference is that the adversary builds up two parallel and intertwined executions *incrementally*, by running many sequential executions. In each step (i.e., protocol execution), an additional message for the parallel execution is generated. This strategy works because the protocol is deterministic. Therefore, it is possible to start an execution from scratch and send the same series of messages (until a certain point), and the honest parties' responses will be identical to their responses in previous executions. Notice that the above strategy works even if the setup phase is probabilistic. This is due to the fact that the setup information is fixed for all executions. Therefore, additional protocol executions relate to the same setup information, as required.

Assume by contradiction that there exists a deterministic protocol for authenticated Byzantine Generals Π for three parties with the following properties: Π tolerates one corrupted party, always halts after at most r rounds, and is secure for $2r$ sequential executions. We show that the existence of such a protocol results in a contradiction. Exactly as in the proof of Theorem 1, we combine two copies of Π into a hexagonal system S with parties G_0, A_0 and B_0 (where G_0 has input 0), and G_1, A_1 and B_1 (where G_1 has input 1). We begin by proving the following claim (that implies an analogue to Claim 3.2).

Claim 4.1 *There exists an adversary $\mathcal{B}$ controlling party B such that in the $2r^{\text{th}}$ sequential execution of Π, the views of G and A (upon input 0 into that execution) are* identical *to the respective views of G_0 and A_0 after r rounds of the system S.*

Proof: In the proof of Theorem 1, adversary $\mathcal{B}$ works by simulating the hexagonal system S. Essentially, this consists of simulating two parallel executions of Π (with the edges "redirected" to make up S); recall that this involves intertwining *two* different copies of Π (one in which G inputs 0, and one in which it inputs 1). However, in our setting, Π can only be run sequentially, and thus only *one* copy of Π can be running at any given time. But we utilize the sequential executions to build the simulation of the hexagon step by step, i.e. round i of the hexagon will be simulated by the $2i$–1^{th} and $2i^{\text{th}}$ executions of Π. More specifically, we will run the first two sequential executions Π_1 and Π_2, and from those two only extract the messages of the first round (and disregard the rest of the computation). This is exactly equivalent to the first step in the parallel case where we took the messages of the parties in the first round, there we simple did the in one time unit as the protocols were running in parallel. Now, we move to Π_3 and Π_4. We will initiate these protocols with the messages that were extract from Π_1 and Π_2 and redirect them as necessary (again exactly as in the parallel case). Here too this process will first happen in Π_3 and afterwards in Π_4. We let both these protocols run for two rounds and extract the second round messages (ignoring the rest of the execution). And we proceed in this manner initiating each invocation with the same prefix as its predecessor (e.g. the predecessor of Π_i is Π_{i-2}) and adding the new messages that were extracted from the predecessors" invocations (e.g. for Π_{2i+1} from Π_{2i-1} and Π_{2i}). This process continues until we have generated the needed messages for each round of the protocol. This is a hybrid process for generating a protocol that is in some sense a composition of two executions, but as we cannot run them in parallel to immediately transfer messages from one to the other this needs to be done in rounds. We now show how this simulation is achieved.

Let G_0 and G_1 be identical copies of G, except that G_0 has input 0 and G_1 has input 1; likewise define A_0, A_1, B_0 and B_1 to be identical copies of A and B, respectively. The sequential executions are such that G_0, A_0 and B_0 run in the $2i^{\text{th}}$ execution; and G_1, A_1 and B_1 run in

the $2i-1^{\text{th}}$ execution, for $1 \leq i \leq r$. The adversarial party $\mathcal{B}$ controls B_0 and B_1, but as in the proof of Theorem 1, $\mathcal{B}$'s malicious behavior consists merely of redirecting messages. We denote by $\mathsf{msg}_\ell(P_a, P_b)$ the ℓ^{th} message sent by party P_a to party P_b in an execution of Π. Furthermore, i denotes the index of the *round* taking place in the system S, and j is the index of the *execution* in the series of sequential executions. Adversary $\mathcal{B}$ works as follows:

1. $i = 1$ (simulation of the 1^{st} round of S):

   (a) $j = 2i - 1 = 1$: $\mathcal{B}$ invokes B_1 who runs with G_1 and A_1. Party $\mathcal{B}$ records the messages output by B_1: $\mathsf{msg}_{i=1}(B_1, G_1)$ and $\mathsf{msg}_{i=1}(B_1, A_1)$ (we stress that B_1 is run internally by $\mathcal{B}$ and therefore these messages are obtained internally). Furthermore, $\mathcal{B}$ receives and records the incoming messages: $\mathsf{msg}_{i=1}(G_1, B_1)$ and $\mathsf{msg}_{i=1}(A_1, B_1)$ (these messages are received by $\mathcal{B}$ through external interaction with G_1 and A_1). Finally, $\mathcal{B}$ runs the execution until it concludes, ignoring the continuation.

   (b) $j = 2i = 2$: $\mathcal{B}$ invokes B_0 who runs with G_0 and A_0. Party $\mathcal{B}$ records the messages output by B_0: $\mathsf{msg}_{i=1}(B_0, G_0)$ and $\mathsf{msg}_{i=1}(B_0, A_0)$. Next, $\mathcal{B}$ receives and records the incoming messages: $\mathsf{msg}_{i=1}(G_0, B_0)$ and $\mathsf{msg}_{i=1}(A_0, B_0)$. Finally, $\mathcal{B}$ runs the execution until it concludes, ignoring the continuation.

2. $i = 2$ (simulation of the 2^{nd} round of S):

   (a) $j = 2i - 1$: $\mathcal{B}$ invokes B_1 who runs with G_1 and A_1. Party $\mathcal{B}$ works as follows (we note that all the messages sent in this round were obtained in the first step of executions 1 and 2):

   - $\mathcal{B}$ passes B_1 the messages $\mathsf{msg}_{i=1}(G_1, B_1)$ and $\mathsf{msg}_{i=1}(A_0, B_0)$ (informally speaking, this second message is redirected).
   - $\mathcal{B}$ sends G_1 the message $\mathsf{msg}_{i=1}(B_1, G_1)$.
   - $\mathcal{B}$ sends A_1 the message $\mathsf{msg}_{i=1}(B_0, A_0)$ (this message is redirected).

   In addition $\mathcal{B}$ receives and records the following messages from the second round of this execution:

   $$\mathsf{msg}_{i=2}(B_1, G_1),\ \mathsf{msg}_{i=2}(B_1, A_1),\ \mathsf{msg}_{i=2}(G_1, B_1) \text{ and } \mathsf{msg}_{i=2}(A_1, B_1)$$

   Finally, $\mathcal{B}$ runs the execution until it concludes, ignoring the continuation. We denote all the messages sent up to this point by Π_j.

   (b) $j = 2i$: $\mathcal{B}$ invokes B_0 who runs with G_0 and A_0. Party $\mathcal{B}$ works as follows:

   - $\mathcal{B}$ passes B_0 the messages $\mathsf{msg}_{i=1}(G_0, B_0)$ and $\mathsf{msg}_{i=1}(A_1, B_1)$.
   - $\mathcal{B}$ sends G_0 the message $\mathsf{msg}_{i=1}(B_0, G_0)$.
   - $\mathcal{B}$ sends A_0 the message $\mathsf{msg}_{i=1}(B_1, A_1)$.

   In addition $\mathcal{B}$ receives and records the following messages from the second round of this execution:

   $$\mathsf{msg}_{i=2}(B_0, G_0),\ \mathsf{msg}_{i=2}(B_0, A_0),\ \mathsf{msg}_{i=2}(G_0, B_0) \text{ and } \mathsf{msg}_{i=2}(A_0, B_0)$$

   Finally, $\mathcal{B}$ runs the execution until it concludes, ignoring the continuation. We denote all the messages sent up to this point by Π_j.

3. $3 \leq i \leq r$ (simulation of the i^{th} round of S):

(a) $j = 2i - 1$: $\mathcal{B}$ invokes B_1 who runs with G_1 and A_1. Party $\mathcal{B}$ works as follows:

- $\mathcal{B}$ runs Π_{j-2}
- $\mathcal{B}$ passes B_1 the messages $\mathsf{msg}_{i-1}(G_1, B_1)$ and $\mathsf{msg}_{i-1}(A_0, B_0)$.
- $\mathcal{B}$ sends G_1 the message $\mathsf{msg}_{i-1}(B_1, G_1)$.
- $\mathcal{B}$ sends A_1 the message $\mathsf{msg}_{i-1}(B_0, A_0)$.

In addition $\mathcal{B}$ receives and records the following messages:

$$\mathsf{msg}_i(B_1, G_1),\ \mathsf{msg}_i(B_1, A_1),\ \mathsf{msg}_i(G_1, B_1) \text{ and } \mathsf{msg}_i(A_1, B_1)$$

Finally, $\mathcal{B}$ runs the execution until it concludes, ignoring the continuation.

(b) $j = 2i$: $\mathcal{B}$ invokes B_0 who runs with G_0 and A_0. Party $\mathcal{B}$ works as follows:

- $\mathcal{B}$ runs Π_{j-2}
- $\mathcal{B}$ passes B_0 the messages $\mathsf{msg}_{i-1}(G_0, B_0)$ and $\mathsf{msg}_1(A_1, B_1)$.
- $\mathcal{B}$ sends G_0 the message $\mathsf{msg}_{i-1}(B_0, G_0)$.
- $\mathcal{B}$ sends A_0 the message $\mathsf{msg}_{i-1}(B_1, A_1)$.

In addition $\mathcal{B}$ receives and records the following messages:

$$\mathsf{msg}_i(B_0, G_0),\ \mathsf{msg}_i(B_0, A_0),\ \mathsf{msg}_i(G_0, B_0) \text{ and } \mathsf{msg}_i(A_0, B_0)$$

Finally, $\mathcal{B}$ runs the execution until it concludes, ignoring the continuation.

First, note that at the conclusion of round 2 of the 4^{th} execution ($j = 4$), the views of parties G_0 and A_0 are identical to their views at the conclusion of round 2 of S (in particular, G_0 sees messages from A_0 and B_0, and A_0 sees messages from G_0 and B_1). Then, in the sixth sequential execution, B begins by sending G_0 and A_0 the same round 2 messages. Since Π is a deterministic protocol, G_0 and A_0 reply with the *same* messages as in the fourth execution (likewise, the messages they send to each other are the same as in the fourth execution). Thus, the round 3 messages that they receive (that are computed based on the messages sent in previous executions) are consistent with their views in S. Using the same argument, we have that for every i, after i rounds of the $2i^{\text{th}}$ sequential execution, the views of G_0 and A_0 are identical to their views after i rounds of S. This concludes the proof of the claim. ■

Recall that by the contradicting assumption, Π is a protocol for Byzantine Generals that always halts within r rounds and is secure for $2r$ sequential executions. Thus, in the $2r^{\text{th}}$ sequential execution of Π, we have that G and A both halt within r rounds and output 0 (their output must equal 0 as G's input is 0). By Claim 4.1, it follows that in S, parties G_0 and A_0 also halt within r rounds and both output 0. Thus, we obtain the following analogue to Claim 3.2:

Claim 4.2 *Except with negligible probability, parties G_0 and A_0 halt within r rounds and output 0 in the system S.*

The following two claims (analogous to Claims 3.3 and 3.4) can be shown in a similar fashion:

Claim 4.3 *Except with negligible probability, parties G_1 and B_1 halt within r rounds and output 1 in the system S.*

Claim 4.4 *Except with negligible probability, parties A_0 and B_1 halt within r rounds and output the same value in the system S.*

Combining Claims 4.2, 4.3 and 4.4, we reach a contradiction. We thus conclude that there does not exist a deterministic protocol for authenticated Byzantine Generals for three parties that tolerates one corrupted party, runs for at most r rounds and is secure for $2r$ sequential executions. As in the proof of Theorem 1, the general case of n parties (for any n) is obtained in a standard way. This completes the proof of the theorem. ■

5 Sequentially Composable Randomized Protocols

In this section we present three results. The first one is a *randomized* protocol for authenticated Byzantine Generals for three parties that can tolerate any number of corrupted parties and remains secure under sequential composition. This in return enables us to achieve our second and third results. The second result is a protocol that tolerates any $t < n/2$ corrupted parties and has polynomial complexity. The third one is a protocol that can tolerate any number of corrupted parties, but its communication complexity is exponential in the number of participating parties. Both protocols are proven secure against adaptive, polynomial-time adversaries, and are secure under *sequential composition*.

We start by describing our first result. Denote such a protocol by $\mathsf{ABG}_{3,3}$. Recall that $\mathsf{ABG}_{n,t}$ denotes an authenticated Byzantine Generals protocol for n parties that tolerates up to t corruptions. We first present a protocol for $\mathsf{ABG}_{3,3}$ in Section 5.1 and prove that it composes sequentially. Then show how it can be used to obtain protocols for general n in Sections 5.2 and 5.3.

The trusted preprocessing phase. The trusted preprocessing phase defined in Definition 3 is very strong. First, the trusted party chooses all of the setup strings by itself. Second, the process of sampling the setup strings need not be efficient. Defining the setup in this way strengthens our impossibility results. However, if such a strong setup was utilized for obtaining a secure protocol (especially, if the setup takes exponential time), then the protocol would not be of much use. We stress that our protocols do *not* require exponential setup, and involve simply choosing signing and verification keys for a secure signature scheme. Furthermore, our protocols even remain secure if each party chooses its own signing and verification key-pair, and publicizes its verification-key in an "ideal way". (For example, it suffices for each party to place its verification-key on a common bulletin board that can be accessed by all other parties. The only property required is that a single verification key is identified with a given party. We note that it is even possible to allow the adversary to see the honest party's keys before it chooses its own.)

5.1 Sequentially Composable $\mathsf{ABG}_{3,3}$

We construct a (randomized) protocol $\mathsf{ABG}_{3,3}$ for three parties that remains secure under sequential composition; denote the parties by G (the General) and P_1, P_2 (the recipients), and denote the General's input value by x. According to Definition 1, parties P_1 and P_2 need to output the same value x' (this is the agreement requirement), and furthermore, if G is not corrupt then it must hold that $x' = x$ (this is the validity requirement). Termination is also required; however, this is trivially fulfilled by all our protocols and we therefore ignore it from here on. As is evident from the proof of the lower bound in Section 4, the central problem in obtaining security in the sequential case is that corrupted parties can *import* signed messages from previous executions, and it is impossible

to distinguish between these "old" messages and the current signatures. Thus, if some freshness associated with the current execution could be introduced into the signatures, this would foil the adversary's actions. This seems to place us in a circular argument, because agreeing on such freshness requires "agreement". 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 only the parties who are *receiving* the signature jointly agree on a fresh string. Fortunately, *two* parties can easily agree on a new fresh value: they simply exchange random messages and each set the fresh string to equal the concatenation of the exchanged values. These strings are then sent to the third party. If the sending parties are honest then the third party receives the same string from both, but if they are not then the third party may receive two different strings. Now, in the protocol that follows whenever the third party signs a message that it sends to the other two parties, it uses the two (fresh) strings which it received from the two parties, i.e. the two strings which were generated in the fashion mentioned before. We note that in the protocol, only the General G signs messages, 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 a signature $\sigma = S_{sk}(m)$ on m also contains the value m.

Protocol 3: Authenticated Byzantine Generals for Three Parties

Public input: Security parameter 1^k
Public verification key vk associated with G

Private input of G: Secret signing key sk corresponding to vk
A value x to be broadcast

The protocol:

1. P_1 and P_2 agree on a random label ℓ, as follows:
   (a) P_1 and P_2 choose random k-bit strings u_1 and u_2, respectively, and send them to each other.
   (b) Each party sets $\ell = u_1 \circ u_2$, 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 computes a signature $\sigma = S_{sk}(x, \ell_1, \ell_2)$ and sends σ to P_1 and P_2.
4. P_1 and P_2 send each other the signatures σ that they received in the previous round.

Output: Let σ and σ' denote the signatures that party P_i received above (i.e., one from G and one from P_j). Then, a value x is said to be valid if σ or σ' constitutes a *valid signature* on (x, ℓ_1, ℓ_2) and $\ell \in \{\ell_1, \ell_2\}$ (i.e., $V_{vk}((x, \ell_1, \ell_2), \sigma) = 1$ and at least one of ℓ_1, ℓ_2 equals the label ℓ generated in Step 1). The output is computed as follows:

1. If a recipient P_i received exactly one valid value x, then it outputs x. Otherwise, it outputs a default value, say 0.
2. The General G always outputs x.

Figure 3: $\mathsf{ABG}_{3,3}$

We now prove that Protocol 3 (in Figure 3) constitutes a secure protocol for $\mathsf{ABG}_{3,3}$ that remains secure under sequential composition. Actually, since Protocol 3 will be used later in a setting with n

parties, we state a broader claim regarding its security under composition, rather than just proving security under sequential composition in the three party setting. Specifically, we consider a network with n parties, where any subset of 3 parties may run any given protocol execution.

Lemma 5.1 *Assume that the signature scheme (Gen, S, V) is existentially secure against adaptive chosen-message attacks. Then, Protocol 3 is a protocol for $\mathsf{ABG}_{3,3}$ that remains secure under sequential composition in the following setting. We assume a system of n parties, in which any $t \leq n$ may be adaptively corrupted. Each execution of the protocol can be carried out by any subset of three parties from the system, where any subset of these three parties can be corrupted.*

Proof: We prove the theorem by contradiction. Assume that there exists an adaptive polynomial-time adversary $\mathcal{A}$ who invokes many sequential executions of Protocol 3 and succeeds in "breaking" at least one of the executions with non-negligible probability. Given this adversary $\mathcal{A}$, we will construct a forger $\mathcal{F}$ who succeeds in breaking the signature scheme (i.e., generating a forgery) with non-negligible probability. This therefore contradicts the security of the signature scheme. Our proof refers to the agreement and validity requirements, as stated in Definition 1 (termination follows immediately from the protocol description).

In our setting, there are n parties, any number of which may be corrupted by the adversary. Therefore, any individual execution of Protocol 3 may involve 0, 1, 2 or 3 corrupted parties. First, if all parties are honest, then the agreement and validity requirements clearly hold. On the other extreme, if all the parties are corrupted, then there are no requirements on the protocol and therefore security holds by default. We next show that if two parties are corrupted and so only a single party is honest, then security holds easily. Specifically, agreement holds vacuously no matter what the honest party outputs. Regarding the validity requirement: If the General G is the (single) honest party, then by Protocol 3, it always outputs x and so validity holds. On the other hand, if G is corrupted, then there is no validity requirement. The above arguments hold irrespective of the number of executions of the protocol. We therefore conclude that the agreement and validity requirements hold for all executions of Protocol 3 where 0, 2 or 3 parties are corrupted.

It follows that the only case in which either the agreement or validity requirements can be foiled is where exactly one party is corrupted (and the other two are honest). We first prove that when the General G is corrupted, then agreement holds (recall that there is no validity requirement in this case). In this case both, P_1 and P_2 are honest, thus they both hold the same label ℓ and both see the same signatures σ and σ' (recall that they send each other these signatures in Step 4). Therefore, a value x is valid for P_1 if and only if it is valid for P_2. This implies that they both either output the same x or the default value 0. As above, this argument holds irrespective of the number of executions (and is also not dependent on the security of the signature scheme).

It remains to be proven that the agreement and validity properties hold when the General G and one of the recipients P_1 or P_2 are honest. In this case, agreement implies validity because G always outputs its input value x. Therefore, if agreement is fulfilled, then the honest recipient must also output x, thereby fulfilling the validity requirement. To prove that agreement hold, we show that if an adversary $\mathcal{A}$ can foil the agreement in an execution where the General is honest, then we can construct a forger $\mathcal{F}$ for the signature scheme (Gen, S, V). The basic idea for the proof is to first observe that G always outputs x and transfers a valid signature on x to the honest receiver. Thus, if the agreement property was foiled, it must be the case that the receiver has received another valid signature on a different value x'. This can happen if the random string generated by the honest party was already used in the past (which is highly unlikely) or that the faulty party has managed in forging a signature under G's signature key, which is a violation of the security properties of the signature scheme. We follow this idea to build a forger. The forger $\mathcal{F}$ receives a

public verification-key vk as input, and is given access to a signing oracle $S_{sk}(\cdot)$ associated with this key. $\mathcal{F}$ begins by choosing one of the parties at random, say P_j, and associates the verification-key vk with this party. Intuitively, with probability $1/n$, this is the party that 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. $\mathcal{F}$ gives the adversary $\mathcal{A}$ all of the public verification keys and, in addition, the secret signing keys of all the initially corrupted parties. Then, $\mathcal{F}$ internally 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 that it initially associated with P_l.
- 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 vk (which is P_j's public verification-key). Therefore, $\mathcal{F}$ computes 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 $\sigma = S_{sk}(x, \ell_1, \ell_2)$.
- *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 in an execution where exactly one party is corrupted. Since by assumption $P_j = G$ is not corrupted, we have that one of the recipients P_i or $P_{i'}$ is corrupted. For the sake of clarity, below we will refer to the parties as G, P_1 and P_2 where, without loss of generality, P_1 is the corrupted party. We note that $\mathcal{F}$ plays the roles of both honest parties G and P_2 in the simulation. Now, since the agreement was foiled, we know that P_2 does not output G's input value x, which means that it outputted some value $x' \neq x$. However, since G is honest, x is clearly a valid value for P_2. Therefore, it must be that the value x' outputted by P_2 is the default value, and P_2 received two valid values x and x'. This implies that P_2 received two different correct signatures that include the label ℓ generated in this execution. However, $\mathcal{F}$ only accessed its signing oracle once in this execution (for signing on the value x, including the labels). Therefore, $\mathcal{F}$ has obtained a valid signature on a value that was not queried to its oracle in this execution. It remains to show that this value was also not queried in any previous execution. However, this follows from the fact that the label ℓ is included in this "forgery", and this label includes random k-bit strings that were generated in the current execution. Therefore, except with negligible probability, the label ℓ did not appear in any previous execution. We conclude that $\mathcal{F}$

obtained a valid signature on a value that was not queried to the signing oracle at any stage. This signature therefore constitutes a successful forgery, as required.

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 $\mathcal{A}$'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. We note that the fact that P_j is chosen ahead of time makes no difference because the simulation is perfect (in particular, using the signing oracle $S_{sk}(\cdot)$ is exactly the same as generating the signatures using sk). 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 the agreement, which is non-negligible. This contradicts the security of the signature scheme. ■

5.2 Sequentially Composable $\mathsf{ABG}_{n,n/2}$

In this section, we use Protocol 3 in order to obtain a protocol that remains secure under sequential composition, and tolerates $t < n/2$ corruptions. We recall that Fitzi and Maurer [17] present a protocol for the Byzantine Generals problem that tolerates any $t < n/2$ corrupted parties, using an ideal 3-party broadcast channel. Their protocol is in the standard (unauthenticated) model and makes no complexity assumptions (i.e., it is in the information-theoretic model). The 3-party broadcast channel that they use enables them to bypass the lower bound of $t < n/3$ [31, 28]). As we have shown in Section 5.1, given a public-key infrastructure for signature schemes, it is possible to implement secure broadcast among three parties that remains secure under sequential composition. Thus, a protocol for $\mathsf{ABG}_{n,n/2}$ is derived by substituting the ideal 3-party broadcast primitive in the protocol of Fitzi and Maurer [17] with Protocol 3. Since Protocol 3 and the protocol of Fitzi and Maurer [17] both remain secure under sequential composition, the same is true of the resulting protocol.

Theorem 4 *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 $t < n/2$ adaptive corruptions and remains secure under sequential composition.*

Proof: As described above, a protocol $\mathsf{ABG}_{n,n/2}$ that remains secure under sequential composition can be constructed by combining the $\mathsf{BG}_{n,n/2}$ protocol of [17] with Protocol 3 for $\mathsf{ABG}_{3,3}$ (recall that by Lemma 5.1, Protocol 3 remains secure under sequential composition played by various 3-party subsets of the n parties). Specifically, each time that parties communicate using the ideal 3-party broadcast channel in the protocol of [17], this communication is replaced by an execution of Protocol 3. All the executions of $\mathsf{ABG}_{3,3}$ can be made sequential by setting a fixed schedule for these executions in the protocol specification. We call this the combined protocol. Intuitively, since the Protocol 3 is secure under sequential composition, it simulates the ideal 3-party broadcast channel, except with negligible probability. Now, by Proposition 2.1, the protocol of [17] remains secure under concurrent (and thus sequential) composition, when using an ideal 3-party broadcast. Therefore, the combined protocol using only a standard point-to-point network must also remain secure under sequential composition. (Note that this is not immediate because Protocol 3 has only been proven secure when it is the only protocol being run sequentially, whereas here it is run together with the protocol of [17]. Nevertheless, the formal proof below works by showing how it is possible to simulate an execution of the protocol [17] while attacking sequential executions of Protocol 3 only.)

Formally, we show that if the combined protocol for $\mathsf{ABG}_{n,n/2}$ can be broken, then this yields an adversary that can break either the protocol of [17] under sequential composition, or Protocol 3

under sequential composition. That is, assume by contradiction that there exists an adversary $\mathcal{A}$ running many sequential executions of the combined protocol, such that with non-negligible probability, $\mathcal{A}$ causes either the validity or agreement requirements to not hold in at least one execution of the combined protocol. Then, there are two possibilities:

1. *All of the executions of Protocol 3 terminate successfully, except with negligible probability:* In this case, we construct an adversary $\mathcal{A}'$ for the protocol of [17] that uses an ideal 3-party broadcast between all triples of parties. The adversary $\mathcal{A}'$ internally simulates for $\mathcal{A}$ the messages sent by the honest parties in the executions of Protocol 3. Specifically, if an honest party broadcasts a message x on the 3-party broadcast channel in the execution of the protocol of [17], then $\mathcal{A}'$ simulates that party broadcasting x using Protocol 3. Likewise, when $\mathcal{A}$ broadcasts a value using Protocol 3 in the internal simulation of the combined protocol, then $\mathcal{A}'$ plays the honest recipients in this execution. At the conclusion of the simulation of this execution of Protocol 3, adversary $\mathcal{A}'$ playing the honest recipients receives a value x'. (Note that both recipients are guaranteed to receive the same x'.) $\mathcal{A}'$ then broadcasts x' to the appropriate recipients on the ideal 3-party broadcast channel. We note that $\mathcal{A}'$ can deal with adaptive corruptions during the simulation, because the honest parties in Protocol 3 have no secret information.

   Notice first that every execution of Protocol 3 that terminates "successfully" (i.e., where validity and agreement hold) perfectly simulates an ideal 3-party broadcast. Therefore, since Protocol 3 is correct except with negligible probability, the above simulation by $\mathcal{A}'$ is perfect except with negligible probability. This implies that the probability that $\mathcal{A}'$ causes either the validity or agreement requirements to not hold in at least one execution of the protocol of [17] is at most negligibly far from the probability that $\mathcal{A}$ causes either the validity or agreement requirements to not hold in at least one execution of the combined protocol. Thus, by the contradicting assumption, $\mathcal{A}'$ "breaks" the protocol of [17] with non-negligible probability, within the setting of sequential composition. This contradicts the security of [17] (recall that by Proposition 2.1, stand-alone security of unauthenticated Byzantine Generals implies security under concurrent, and thus sequential, composition).

2. *With non-negligible probability, at least one of the executions of Protocol 3 terminates such that validity or agreement does not hold:* This does not immediately yield a contradiction because $\mathcal{A}$ "breaks" one of the executions of Protocol 3, when being run as a subprotocol within the protocol of [17]. In contrast, the security of Protocol 3 has only been proven within the context of sequential composition, where it is the only protocol being run by the parties. Nevertheless, an adversary $\mathcal{A}'$ can be constructed for the sequential composition of Protocol 3 in a straightforward way, as follows. $\mathcal{A}'$ internally invokes $\mathcal{A}$ and simulates all of the honest parties for the messages of the protocol of [17]. Furthermore, when a party is supposed to broadcast a value x in this simulation, then $\mathcal{A}'$ sets the appropriate party's input for Protocol 3 to x. (Recall that in the setting of sequential composition, the adversary can set the parties' inputs for every execution; see Definition 4). Then, $\mathcal{A}'$ forwards the messages in this execution of Protocol 3 between $\mathcal{A}$ and the honest participating parties. This perfectly simulates an execution of the combined protocol for $\mathcal{A}$. Furthermore, the honest parties run the sequential executions of Protocol 3 in the same way here as in a real execution of the combined protocol, due to the sequential manner in which Protocol 3 substitutions were introduced into the combined protocol (see the beginning of the proof). Therefore, $\mathcal{A}'$ breaks one of the executions of Protocol 3 in this setting of sequential composition with the *same*

probability that $\mathcal{A}$ breaks an execution of Protocol 3 within the combined protocol. That is, $\mathcal{A}'$ breaks Protocol 3 with non-negligible probability, in contradiction to Lemma 5.1.

Both above possibilities results in a contradiction. We therefore conclude that the combined protocol is a protocol for $\mathsf{ABG}_{n,n/2}$ that remains secure under sequential composition. ■

Round complexity. The protocol described in the proof of Theorem 4 has a higher round complexity than the original protocol of [17]. This is due to the fact that the executions of Protocol 3 must be run sequentially, whereas the 3-party ideal broadcast channel of [17] can be used in parallel. Nevertheless, using the methodology of Section 6, it is possible to reduce the round complexity to be of the same order as the original protocol of [17]. The idea is that the protocol specification can number each execution, thus providing unique identifiers. Then, as shown in Section 6, this enables the executions to be run securely in parallel or even concurrently, thus reducing the round complexity. Of course, the same identifiers will be reused in different executions of the overall protocol, but these ones are run sequentially. Thus, the protocol still remains secure under *sequential* composition because the addition of the identifiers requires only small modifications to the proof of Lemma 5.1.

5.3 Sequentially Composable $\mathsf{ABG}_{n,n}$, (for small n)

In this section we describe a protocol for the Byzantine Generals problem for n parties, that can tolerate *any number of corrupted parties*. However, the protocol complexity (specifically, the number of messages sent) is exponential in the number of participating parties (actually, for n parties the protocol uses $O(2^n n!)$ messages). Therefore, this protocol can only be efficiently carried out for $n = \log k / \log\log k$ parties, where k is the security parameter. We stress that this limitation on the number of parties is due to two reasons. First, we wish the protocol to run in time that is polynomial in the security parameter k. 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 protocol $\mathsf{ABG}_{n-1,n-1}$ (for $n-1$ parties) that (tolerates any number of corrupted parties and) remains secure under sequential composition, and produces a protocol $\mathsf{ABG}_{n,n}$ that remains secure under sequential composition. This transformation can then be iteratively applied to Protocol 3 for $\mathsf{ABG}_{3,3}$ in order to obtain a protocol $\mathsf{ABG}_{n,n}$, for any n.

The idea for the transformation is closely related to the ideas behind the protocol for Byzantine Generals for three parties. Recall that our 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 point-to-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 $\mathsf{ABG}_{n-1,n-1}$ protocol to have all parties (apart from the General) agree on a fresh label. Then, each party privately sends this label to the General, who then includes all labels in its signatures. Finally, each party uses $\mathsf{ABG}_{n-1,n-1}$ again to notify all other parties (except the General) of the signature that it claims to have received from the General. Thus, we prove:

Theorem 7 *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 an authenticated Byzantine Generals protocol for* $O(\log k / \log\log k)$ *parties, that tolerates any number of corrupted parties and remains secure under sequential composition.*

Proof: As described above, Theorem 7 is proven by providing a transformation of any protocol for $\mathsf{ABG}_{n-1,n-1}$ that remains secure under sequential composition into a protocol for $\mathsf{ABG}_{n,n}$ that remains secure under sequential composition. The transformation is then applied to Protocol 3 for $\mathsf{ABG}_{3,3}$, yielding the desired result. The reason that security is obtained for only $O(\log k/\log\log k)$ parties is due to the complexity of the final protocol, as will be shown later. We begin by presenting the transformation in Figure 4.

Protocol 4: Authenticated Byzantine Generals for n Parties

Public input: Security parameter 1^k
Public keys $vk_1, \ldots, vk_n$ associated with parties $P_1, \ldots, P_n$

Private input of P_i: Secret key sk_i corresponding to vk_i
The General $G = P_n$ also has a value x to be broadcast

The protocol:

1. $P_1, ..., P_{n-1}$ agree on a random label ℓ, as follows:
   (a) For every i $(1 \leq i \leq n-1)$, P_i chooses a random k-bit string u_i and plays the General in an execution of $\mathsf{ABG}_{n-1,n-1}$ in order to broadcast u_i to the rest of the recipients (i.e., to all $P_j \neq P_n$). All these executions are run sequentially.[7]
   (b) Each party sets $\ell = \circ_{i=1}^{n-1} u_i$, where $\circ$ denotes concatenation.
2. Each P_i $(1 \leq i \leq n-1)$ sends ℓ to G.
3. Let ℓ_i denote the message that G received from P_i in the previous round. G computes a signature $\sigma S_{sk_G}(x, \ell_1, ..., \ell_{n-1})$ and sends σ to all parties $P_1, \ldots, P_{n-1}$.
4. Let σ_i denote the message that P_i received from G. Then, for every i $(1 \leq i \leq n-1)$, P_i plays the General in an execution of $\mathsf{ABG}_{n-1,n-1}$ in order to broadcast σ_i to the rest of the recipients (i.e., to all $P_j \neq P_n$). All these executions are run sequentially.[7]

Output: Let $\sigma_1, \ldots, \sigma_{n-1}$ denote the signatures that party P_i received above (i.e., σ_i from G and all other signatures σ_j from parties P_j). Then, a value x is said to be valid if there exists a $j \in \{1, \ldots, n-1\}$ such that the signature σ_j constitutes a *valid signature* on a string beginning with x and including the label ℓ, as generated above in Step 1. Output is computed as follows:

1. If a recipient P_i received exactly one valid value x, then it outputs x. Otherwise, it outputs a default value, say 0.
2. The General G always outputs x.

Figure 4: Transformation from $\mathsf{ABG}_{n-1,n-1}$ to $\mathsf{ABG}_{n,n}$

Proposition 5.2 *Assume that $\mathsf{ABG}_{n-1,n-1}$ is a Byzantine Generals protocol for $n-1$ parties that tolerates any number of corrupted parties and remains secure under sequential composition. Then, Protocol 4 is an authenticated Byzantine Generals protocol for n parties that tolerates any number of corrupted parties and remains secure under sequential composition.*

Proof (sketch): The proof of this theorem is very similar to the proof of Lemma 5.1; we therefore

[7] These executions can actually be run in parallel using the methodology described in Section 6, in a similar way to that described at the end of Section 5.2.

present only a sketch. We distinguish between executions where the General G is corrupt and executions where it is honest:

G is corrupt: In this case we only need to prove that all honest parties will output the same value(i.e. agreement . This follows directly from the correctness of the $\mathsf{ABG}_{n-1,n-1}$ protocol. Specifically, the use of $\mathsf{ABG}_{n-1,n-1}$ to broadcast the u_i's guarantees that all honest parties receive the same u_i values and therefore set the same label ℓ. In addition, by the second use of $\mathsf{ABG}_{n-1,n-1}$ to broadcast the received signatures we have that all honest parties will obtain the same set of signatures $\sigma_1, \ldots, \sigma_{n-1}$. Therefore, if a value x is valid for one honest party, then it is also valid for every other honest party. This implies that all honest parties either output the same value x or the default value. We note that this holds irrespective of how many executions have passed, and therefore also in the setting of sequential composition.

G is honest: In this case, we need to show that all honest parties output G's input value x. As in the previous case, all honest parties set the same label ℓ and all honest parties receive the same signature σ that begins with x and includes the label ℓ. By the security of $\mathsf{ABG}_{n-1,n-1}$, it follows that x is a valid value for all honest parties. However, this does not suffice for proving that the honest parties output x, because it is possible that a corrupted party broadcasts a valid signature σ' that begins with some $x' \neq x$ and includes the label ℓ. This is not possible, except with negligible probability, because it involves forging a signature. Specifically, since the label ℓ includes the random strings u_i sent by the honest parties, it is different from the label used in all previous executions (except with negligible probability). Therefore, any valid signature on x' and ℓ must have been generated by the adversary.

This completes the proof of Proposition 5.2. ■

It remains to analyze the complexity of Protocol 4 for n parties. This can be computed recursively as follows. A single execution of Protocol 4 requires $2(n-1)$ executions of $\mathsf{ABG}_{n-1,n-1}$ for broadcasting all the u_i and σ_i values, plus a fixed amount of work that is polynomial in n and k (we denote this by $\mathrm{poly}(n,k)$). This yields the following formula, where $T(n)$ denotes the complexity of $\mathsf{ABG}_{n,n}$: $T(n) = 2(n-1) \cdot T(n-1) + \mathrm{poly}(n,k)$. (Recall that $T(3)$ is a fixed polynomial in k, as shown in Protocol 3.) Solving this recursion, we obtain that $T(n) = O(2^n \cdot n! \cdot \mathrm{poly}(n,k))$, which is polynomial in k as long as $n = O(\log k / \log\log k)$. This completes the proof of Theorem 7. ■

5.4 Additional Sequentially-Composable Protocols

We obtain additional results by plugging Protocol 4 into the broadcast protocols of [10]. Specifically, in [10] it is shown that given a primitive that achieves broadcast amongst b parties (with any number of corruptions), it is possible to construct a broadcast protocol tolerating $t < n - \frac{2n}{b+1}$ corruptions. Furthermore, their protocol is polynomial as long as $b = O(1)$. Thus, for example, it is possible to use Protocol 4 to achieve broadcast among any 5 parties. Plugging this into [10], we obtain a broadcast protocol that tolerates any $t < \frac{2n}{3}$ number of corruptions. We thus obtain a *polynomial protocol* for any n that tolerates more than $n/2$ corruptions. We remark that by running the executions of Protocol 4 inside the protocol of [10] sequentially, we obtain that the entire protocol remains secure under sequential composition.

We remark that the above is just an example and we can plug in the instantiation of Protocol 4 for 19 parties and obtain tolerance in the face of $t < 9n/10$ corruptions. In general, for any constant ϵ it is possible to achieve tolerance in the presence of $t < (1-\epsilon)n$ corruptions by using Protocol 4 to obtain a protocol for $b = \frac{2}{\epsilon} + 1$ parties and then plugging this into the protocol of [10]. Notice that

since ϵ is constant, so too is b and thus Protocol 4 is polynomial. We therefore have the following theorem:

Theorem 8 *Assume that there exists a signature scheme that is existentially secure against chosen-message attacks. Then, for any constant $0 < \epsilon < 1$ there exists a randomized authenticated Byzantine Generals protocol for any number n parties, that tolerates $t < (1 - \epsilon)n$ corrupted parties and remains secure under sequential composition.*

6 Authenticated Byzantine Generals 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 Generals that composes concurrently, for *any* number of corrupted 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 by the results of this section, agreeing on a common identifier amounts to solving the Byzantine Generals problem under concurrent composition, and we have proven that this cannot be achieved for $t \geq n/3$. 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 impossibility result is based on the ability of the adversary to *borrow* signed messages from one execution to another. Now, if each signature also includes the unique 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 of protocols for authenticated Byzantine Generals to protocols that compose concurrently in a setting where unique identifiers exist. Thus, the simple idea that enables the composition is to include the unique identifier under the signature. Our transformation holds for protocols that utilize the signature scheme for signing and verifying only (as is natural). We formally define our transformation:

The transformation. Let (Gen, S, V) be a secure signature scheme and let id be a string (of any length). We define (Gen, S_{id}, V_{id}) as follows: $S_{id}(sk, m) = S(sk, id \circ m)$ and $V_{id}(vk, m, \sigma)V(vk, id \circ m, \sigma)$, where $\circ$ denotes concatenation. That is, (Gen, S_{id}, V_{id}) are the same as (Gen, S, V) except that the message m is always prefixed by the string id.

Let Π be a protocol for authenticated Byzantine Generals that uses a secure signature scheme (Gen, S, V) for signing and verification of signatures only. We define a modified protocol $\Pi(id)$ that is exactly the same as Π except that the parties use the signature scheme (Gen, S_{id}, V_{id}) as defined above. We note that the common value id is given to each party as auxiliary input.

Let Π be a protocol whose security is based on an existentially secure signature scheme (used for signing only), where the security proof reduction is done without access to a signing oracle. As the protocol $\Pi(id)$ considers signatures as valid only on a subset of the message space, i.e. messages that start with id, we would like to specify the security of $\Pi(id)$ while taking this into account. That is ensuring the security of $\Pi(id)$ while allowing the attacker access to valid signatures on the rest of the message space, i.e. all messages that do not start with the string id. We say the protocol $\Pi(id)$ is *subset secure* if it remains secure when the adversary can query the signature of any message whose prefix is not id.

In the following theorem we show that the above simple transformation and security requirements from the protocol suffice for achieving security in a setting where many concurrent executions take place, as long as each execution has a unique identifier.

Theorem 6 *Restated. Let $\Pi(id)$ be a protocol for authenticated Byzantine Generals, obtained from a protocol Π for ABG as described above, that is subset secure for a single execution when using a signature scheme that is existentially secure against chosen-message attacks. Furthermore, the (honest) parties use their secret keys for signing and verify signatures only. Let $id_1, \ldots, id_\ell$ be a series of ℓ distinct equal-length strings* [8]*. Then the protocols $\Pi(id_1), \ldots, \Pi(id_\ell)$ all solve the Byzantine Generals problem, even when run concurrently.*

Proof: Intuitively, the security of the protocols $\Pi(id_1), \ldots, \Pi(id_\ell)$ is due to the fact that signatures from $\Pi(id_i)$ cannot be of any help to the adversary in $\Pi(id_j)$, as it is subset secure. That is, it remains secure even with access to additional signatures under the same signing key. In the concurrent execution the other protocols will serve as oracles to the signature scheme, but this will not help in breaking the single execution. This proof is almost straight forward from the definition of the subset security of the protocol $\Pi(id)$, and from the fact that in $\Pi(id_j)$, the honest parties reject any signature on a message that begins with an identifier that is not id_j. Since $id_i \neq id_j$, we have that signatures sent in $\Pi(id_i)$ are of no help in $\Pi(id_j)$. Our formal proof of this intuition proceeds by showing how an adversary for a single execution of $\Pi(id)$ can internally simulate the concurrent executions of $\Pi(id_1), \ldots, \Pi(id_\ell)$, thereby reducing the security of the concurrent setting to the stand-alone setting.

Assume that there is an adversary $\mathcal{A}$ who successfully attacks the concurrent executions of $\Pi(id_1), \ldots, \Pi(id_\ell)$. We will use it to build an adversary $\mathcal{A}_{id}$ to successfully attack a single execution of $\Pi(id)$ for some string id. This then contradicts the security of the underlying protocol $\Pi(id)$. Now, assume by contradiction that $\mathcal{A}$ succeeds in "breaking" one of the $\Pi(id_i)$ executions with non-negligible probability. The adversary $\mathcal{A}_{id}$ internally incorporates $\mathcal{A}$ and attacks a single execution of $\Pi(id)$. Intuitively, $\mathcal{A}_{id}$ internally simulates all executions for $\mathcal{A}$, except for the one in which $\mathcal{A}$ succeeds in its attack. Specifically, $\mathcal{A}_{id}$ first uniformly at random selects an execution $i \in \{1, \ldots, \ell\}$ and sets $id = id_i$. Next, $\mathcal{A}_{id}$ invokes $\mathcal{A}$ and emulates the concurrent executions of $\Pi(id_1), \ldots, \Pi(id_\ell)$ for $\mathcal{A}$. Adversary $\mathcal{A}_{id}$ does this by playing the roles of the honest parties in all but the i^{th} execution $\Pi(id_i)$. In contrast, in $\Pi(id_i)$ adversary $\mathcal{A}_{id}$ externally interacts with the honest parties and passes messages between them and $\mathcal{A}$. When $\mathcal{A}_{id}$ needs to produce a signature for a message of an honest party whose prefix is different than id then it access the signature oracle. Note, that $\mathcal{A}_{id}$ never queries the oracle on messages whose prefix is id. Therefore, the emulation by $\mathcal{A}_{id}$ of the concurrent executions for $\mathcal{A}$ is perfect. This implies that $\mathcal{A}_{id}$ succeeds in "breaking" $\Pi(id)$ with success probability that equals $1/\ell$ times $\mathcal{A}$'s success probability in the concurrent setting. Thus, $\mathcal{A}_{id}$ succeeds with non-negligible probability and this contradicts the stand-alone subset security of $\Pi(id)$. ■

Claim 6.1 *Let Π be the secure protocol in (either) [31, 28] for authenticated Byzantine Generals that uses a signature scheme which is existentially secure against chosen message attacks. Let $\Pi(id)$ be a transformation as above. Then $\Pi(id)$ is a subset secure authenticated Byzantine Generals protocol.*

It is easy to verify the above claim by a simple modification of their proofs. We therefore obtain the following theorem:

[8] More generally, any set of ℓ prefix-free strings suffice.

Theorem 9 *Assume that there exists a signature scheme that is existentially secure against adaptive chosen message attacks. Then, in a model where global unique identifiers are allocated to each execution, there exist protocols for authenticated Byzantine Generals that tolerate any $t < n$ corruptions and remain secure under concurrent composition.*

7 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 remain secure under sequential composition, for *any* n and any number of corrupted parties. Second, it is unknown whether or not it is possible to construct a deterministic protocol that terminates in r rounds and remains secure for ℓ sequential executions, 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 corrupted parties.

Acknowledgments

We would like to thank Oded Goldreich for pointing out a simpler proof of Theorem 6, and to the amazing work that he has put in commenting on the write-up. Thanks to Matthias Fitzi for discussions about [17].

References

[1] B. Barak. How to Go Beyond the Black-Box Simulation Barrier. In *42nd FOCS*, pages 106–115, 2001.

[2] B. Barak, Y. Lindell and T. Rabin. A Note on Secure Protocol Initialization and Setup in Concurrent Settings. *Cryptology ePrint Archive,* Report 2004/006, 2004.

[3] D. Beaver. Secure Multi-party Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority. *Journal of Cryptology,* 4(2):75–122, 1991.

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

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

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

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

[8] R. Canetti and T. Rabin. Universal Composition with Joint State. In *CRYPTO 2003*, Springer-Verlag (LNCS 2729), pages 265–281, 2003.

[9] D. Chaum, C. Crepeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. In *Proc. 20th STOC, 1988* pages 11–19.

[10] J. Considine, M. Fitzi, M. Franklin, L.A. Levin, U. Maurer and D. Metcalf. Byzantine Agreement Given Partial Broadcast. *Journal of Cryptology*, 18(3):191–217, 2005.

[11] Y. Dodis and S. Micali. Parallel Reducibility for Information-Theoretically Secure Computation. In *CRYPTO'00*, Springer-Verlag (LNCS 1880), pages 74–92, 2000.

[12] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. In 30*th STOC*, pages 409–418. 1998.

[13] D. Dolev and H.R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal of Computing*, 12(4):656–665, 1983.

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

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

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

[17] M. Fitzi and U. Maurer. From Partial Consistency to Global Broadcast. In 32*th STOC*, pages 494–503. 2000.

[18] J. Garay and P. Mackenzie. Concurrent Oblivious Transfer. In 41*st FOCS*, pages 314–324, 2000.

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

[20] O. Goldreich. Cryptography and Cryptographic Protocols. In *Distributed Computing,* 16(2):177–199, 2003.

[21] O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.

[22] O. Goldreich and A. Kahan. How To Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[23] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM Journal on Computing*, 25(1):169–192, 1996.

[24] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In *Proc.* 19*th STOC*, pages 218–229. ACM, 1987.

[25] S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. *Journal of Cryptology,* 18(3):247–287, 2005.

[26] S. Goldwasser, S. Micali, and R. L. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[27] 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*, pages 139–157, 1995.

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

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

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

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

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

[33] R. Richardson and J. Kilian. On the Concurrent Composition of Zero-Knowledge Proofs. In *EUROCRYPT'99*, Springer-Verlag (LNCS 1592), pages 415–431, 1999.

[34] R.L. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[35] A.C. Yao. Protocols for secure computations. In Proceedings of FOCS'82, pages 160–164, Chicago, 1982. IEEE.