On Composition of Authenticated Byzantine Generals

(Extended Abstract)

Anuj Gupta Prasant Gopal Piyush Bansal Kannan Srinathan
Center for Security, Theory and Algorithmic Research
International Institute of Information Technology, Hyderabad, India
{anujgupta@research. prasant@research. piyush_bansal@research. srinathan@}iiit.ac.in

Abstract

Pease *et al.* introduced the problem of *Authenticated Byzantine Generals* (ABG) where players are augmented with digital signatures to thwart the challenge posed by Byzantine faults in protocols for agreement in a distributed environment. It is well known that ABG among n players tolerating up to any t malicious players is possible if and only if $n > t$, which is a remarkable improvement over the lower bound of $n > 3t$ for same functionality in absence of digital signatures. Subsequently, Lindell *et al.* surprisingly prove that if $n \leq 3t$, there does not exist any protocol for ABG that can compose in parallel even twice (assuming no joint state). However on a more optimistic note, they show that if each run of the protocol is further augmented with a unique session identifier, protocols for ABG which compose in parallel for any number of executions can be designed tolerating $t < n$ faults.

Contrary to the state-of-the-art, we prove that if $n < 2t$, there cannot exist any protocol for ABG, in spite of unique session identifiers, that composes in parallel even twice. Further, for $n \geq 2t$, we design ABG protocols that compose in parallel. From the extant literature one might be encouraged to search for a good and realistic model strictly weaker than authentication with unique session identifiers, where the bound of $n > t$ still holds. Our work shows that any such quest is futile. Rather, our results indicate that in order to attain the bound of $n > t$, one needs a model strictly stronger than that of authentication with unique session identifiers which might only make it more unrealistic.

Keywords: Authenticated Byzantine Generals, protocol composition, unique session identifiers.

1 Introduction

Consider a set of n players $\mathbb{P}=\{p_1, p_2 \ldots p_n\}$ over a completely connected synchronous network. Any protocol in this setting is executed in a sequence of *rounds* where in each round, a player can perform some local computation, send new messages to all the players, receive messages sent to him by other players in the same round, (and if necessary perform some more local computation), in that order. The notion of faults in the system is captured by a virtual entity called *adversary*. During the execution, the (polynomial-time) adversary[1] may take control of up to any t players and make them behave in any arbitrary fashion. Such an adversary is called as a t-adversary. We further assume that the communication channel between any two players is perfectly reliable and authenticated. We also assume existence of a (signature/authentication) scheme via which players authenticate themselves. This is modeled by all the players having an additional setup-tape that is generated during the preprocessing phase. Note that keys cannot be generated with in the system itself. It is assumed that the keys are generated using a trusted system and distributed to players prior to running of the protocol similar to [29]. Typically in such a preprocessing phase, signatures and verification keys are generated. That is, each player gets his own private signature key, and in addition, public verification keys for all the other players. No player can forge any other player's signature and the receiver can uniquely identify the sender of the message using the signature. However, the adversary can forge the signature of all the t players under its control. Further, we assume that each run of a protocol is augmented with *unique session identifiers* (USIDs).

Informally, in an authenticated Byzantine Generals (ABG) protocol the General starts with an input v and the goal is to ensure that after a finite number of rounds of information exchange, all the honest players must consistently output a value u such that if the General is honest, $u = v$. Further, the goal must be met in spite of several parallel executions of the protocol. A brief literature survey is presented in Section 8.

Our Results: The contributions of this paper are three fold: (1) Contrary to the results in extant literature [29, 30], we show that $n > t$ is *not* sufficient for solving ABG under parallel composition even with USIDs. (2) We prove that over a completely connected synchronous graph of n nodes, of which up to any t are controlled by adversary, protocols with USIDs for ABG compose for any number of parallel executions if and only if $n \geq 2t$. (3) Lindell *et al.* [29, 30] raise the question of finding a realistic computation model for ABG that does allow parallel and concurrent composition for n/3 or more corrupted players – our results imply that using authentication with additional power of unique session identifier helps in increasing the fault tolerance but only to an extent, i.e, if one wishes to achieve a tolerance to any number of faults i.e. $n > t$, one needs an even more powerful model than authentication with USIDs which might only make it more "unrealistic".

2 Defining Composable ABG

We directly adopt the definition of [29] to capture the notion of parallel composition of ABG protocols.

Definition 1 (Composable ABG [29]) *Let $p_1, \ldots, p_n$ be players for an ABG protocol Π. Then, Π remains secure under parallel composition if for every polynomial time adversary $\mathcal{A}$, the requirements for ABG (which is elaborated in Definition 2) hold for Π for every execution within the following process: Repeat the following process in parallel until the adversary halts:*

[1]Digital signatures based authentication necessitates the assumption of a polynomial-time adversary. Our impossibility proofs do not need this assumption but our protocols require a "magical" means to authenticate if the adversary is powerful.

1. *The adversary $\mathcal{A}$ chooses the input v for the General Gen.*

2. *All players are invoked for an execution of* Π *(using the strings generated in the preprocessing phase and an* unique session identifier *for this execution). All the messages sent by the corrupted players are determined by the adversary $\mathcal{A}$, whereas all other players follow the instructions of* Π.

Furthermore, as noted by Lindell *et al.*, Definition 1 implies that all honest players are oblivious of the other executions that are taking place in parallel. In contrast, the adversary $\mathcal{A}$ can coordinate between the parallel executions, and the adversary's view at any given time includes all the messages received in all the executions.

We use the well established ideal/real process simulation paradigm to define the requirements of ABG. Both the ideal process and the real process have the set $\mathbb{P}$ of n players including the General *Gen* as common participants. Apart from these, the ideal process has a TTP (trusted third party) and an ideal process adversary $\mathcal{S}$ whereas the real process has a real process adversary $\mathcal{A}$. We start by defining the ideal process for ABG.

Ideal process (Ψ_{ideal}): We assume that all message transmissions in the following protocol are perfectly secure. (1) *Gen* sends his value v to TTP and TTP forwards the same to $\mathcal{S}$. (2) TTP also sends v to all the n players and $\mathcal{S}$. (3) All honest players output v. $\mathcal{S}$ determines the output of faulty players.

Let $IDEAL_{TTP,\mathcal{S}}(v, r_{\mathcal{S}}, \overrightarrow{r})$ denote a vector of outputs of all n players running Ψ_{ideal} where *Gen* has input v, $\mathcal{S}$ has random coins $r_{\mathcal{S}}$ and $\overrightarrow{r} = r_1, r_2 \ldots r_n, r_{TTP}$ are the random coins of n players and the TTP respectively. Let $IDEAL_{TTP,\mathcal{S}}(v)$ denote the random variable describing $IDEAL_{TTP,\mathcal{S}}(v, r_{\mathcal{S}}, \overrightarrow{r})$ when $r_{\mathcal{S}}$ and $\overrightarrow{r}$ are chosen uniformly at random. Let $IDEAL_{TTP,\mathcal{S}}$ denote the ensemble $\{IDEAL_{TTP,\mathcal{S}}(v)\}_{v\in\{0,1\}}$.

Real life process ($\Psi_{real}(\Pi)$): Unlike in the ideal process, here the players interact among themselves as per a designated protocol Π and the real process adversary $\mathcal{A}$. Specifically: (1) Every honest player proceeds according to the protocol code delegated to him as per Π. (2) The adversary $\mathcal{A}$ may send some arbitrary messages (perhaps posing as any of the corrupt players) to some/all of the players. (3) Honest players output a value as per Π. $\mathcal{A}$ determines the output of faulty players.

Let $REAL_{\Pi,\mathcal{A}}(v, r_{\mathcal{A}}, \overrightarrow{r})$ denote a vector of output of all n players running $\Psi_{real}(\Pi)$ where *Gen* has input v, and $r_{\mathcal{A}}, \overrightarrow{r} = r_1, r_2 \ldots r_n$ are the random coins of the adversary and n players respectively. Let $REAL_{\Pi,\mathcal{A}}(v)$ denote the random variable describing $REAL_{\Pi,\mathcal{A}}(v, r_{\mathcal{A}}, \overrightarrow{r})$ when $r_{\mathcal{A}}$ and $\overrightarrow{r}$ are chosen uniformly at random. Let $REAL_{\Pi,\mathcal{A}}$ denote the ensemble $\{REAL_{\Pi,\mathcal{A}}(v)\}_{v\in\{0,1\}}$.

Definition 2 (ABG) *A protocol* Π *is said to be an ABG protocol tolerating a t-adversary if for any subset $I \subset \mathbb{P}$ of cardinality up to t (that is , $|I| \leq t$), it holds that for every probabilistic polynomial-time real process adversary $\mathcal{A}$ that corrupts the players in I in $\Psi_{real}(\Pi)$, there exists a probabilistic polynomial-time ideal process adversary $\mathcal{S}$ in Ψ_{ideal} that corrupts the players in I, such that the ensembles $IDEAL_{TTP,\mathcal{S}}$ and $REAL_{\Pi,\mathcal{A}}$ are computationally indistinguishable.*

3 Corrupting Less Can Damage More!

We now argue that it may not always be in the best interest of the adversary to corrupt players at full-throttle in every concurrent execution. We present two scenarios, where it seems that a protocol may be required to do more work if the adversary chooses not to corrupt same set of players in every concurrent execution. Consider a protocol (with USIDs) over completely connected 3 players $\{a, b, c\}$

tolerating a 2-adversary that solves ABG and composes in parallel twice (existence of such a protocol is well known as $n > t$). W.l.o.g a is the *Gen*.

Consider a scenario s_1: let $\{a, b, c\}$ run two parallel executions of the protocol, say E_1 and E_2. Real process adversary $\mathcal{A}$ corrupts players a, b in both the executions. The *Gen* a starts with input value 0. Consider the corresponding ideal process execution as shown in Figure 1. In ideal execution, player c (encircled) in both E_1 and E_2 is bound to receive correct value from the TTP. The protocol has to ensure that in both E_1 and E_2, player c decides on a correct value (this could be either 0 or 1 as the *Gen* is corrupt).

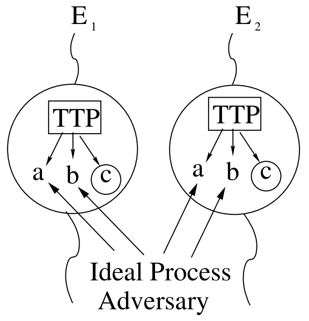

Figure 1: Corresponding ideal process execution for a scenario s_1.

Consider another scenario s_2: $\{a, b, c\}$ run two parallel executions of the protocol, say E_1 and E_2. $\mathcal{A}$ corrupts player a in E_1 and player b in E_2. The *Gen* a starts with value 0 in E_1 and value 1 in E_2. Consider the corresponding ideal process execution as shown in Figure 2. In ideal execution, players b, c in E_1 and players a, c in E_2 receive correct value from the TTP. The protocol has to ensure that players b, c in E_1 decide on same value (this could be either 0 or 1 as the *Gen* is corrupt). However in E_2, protocol must ensure that players a, c decide on value 1. It appears as though in scenario s_2 *the protocol is required to do much more work* as compared to scenario s_1 since scenario s_2 requires different people to agree in different executions.

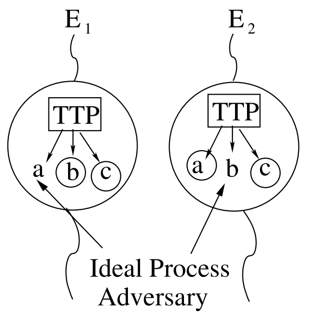

Figure 2: Corresponding ideal process execution for a scenario s_2.

It is conceivable that for scenarios such as s_2, the protocol *may not* ensure correct agreement in each of the parallel executions. In section 4, we prove that there does not exist any protocol(with USID) that composes in parallel twice and solves ABG for $n = 3$, $t = 2$. In the same section we prove that $n > t$ is *not* sufficient for parallel composition of protocols for ABG (with USID). Rather in section 5 we go on to prove a much stronger statement that $n \geq 2t$ *is necessary and sufficient for parallel composition of protocols for ABG (with USID).*

Note that in context of concurrent executions, a t-adversary may corrupt t players in the all the executions or may choose to corrupt less than t players in some parallel execution. That is, the adversary may choose to corrupt say t_1 players ($t_1 < t$) in one execution and another t_2 players (such that $t_1 + t_2 \leq t$) in some other parallel execution[2]. We claim that the proof given in extant literature [29, 30] for sufficiency of $n > t$ for parallel composition of protocols (using USIDs) for ABG implicitly assumes that the adversary *always* corrupts same set of players across *all* parallel executions. In section 7, we formally show that if the above assumption is not true, the proof for sufficiency of $n > t$ in the state-of-the-art breaks down.

4 $n > t$ is not Sufficient for Parallel Composition of ABG

We now formally show that $n > t$ is not sufficient for parallel composition of protocols (using USIDs) solving ABG. We substantiate our claim by proving that there *does not* exist any protocol Π using USIDs that solves ABG and composes in parallel even twice over a completely connected graph

[2]A player running multiple executions can be visualized as a processor running parallel threads. Adversary can attack a player, ask him to execute a code different from the protocol in *some(or all)* of the threads. This is same as adversary corrupting this player in some(or all) executions.

G(Figure 3) of 3 players $\mathbb{P}$=$\{A$,B,$C\}$ influenced by a 2-adversary (2-out-of-3). For the rest of the paper we refer to a protocol Π using USIDs that composes in parallel k times and solves ABG[definition 1] as $\Pi_{k,\ USID}$.

Theorem 1 *There does not exist any $\Pi_{2,\ USID}$ tolerating a 2-adversary over a completely connected graph G of 3 nodes.*

Proof sketch: We assume there exists a protocol $\Pi_{2,\ USID}$ over G tolerating 2-adversary. Our proof essentially demonstrates that there exist two parallel executions of $\Pi_{2,\ USID}$, where the real process adversary $\mathcal{A}$ (t=2) can ensure that honest players in one of the executions do not have a consistent output. In contrast, in the ideal execution honest players are guaranteed to have a consistent output. This implies that there *does not* exist any ideal process adversary $\mathcal{S}$ who can ensure that the output distributions are indistinguishable, thus violating Definition 1.

Using the proof technique developed by Fischer *et al.* [16], we show that $\mathcal{A}$ can ensure that in one of the parallel executions of $\Pi_{2,\ USID}$, honest people exhibit contradictory behavior. Using $\Pi_{2,\ USID}$ we create a protocol π'[Definition 3] in such a way that if $\Pi_{2,\ USID}$ exists then so does π'(Lemma 2). Using two copies of π' we construct a system L (as shown in Figure 3), and show that L must exhibit contradictory behavior. This implies impossibility of the assumed protocol $\Pi_{2,\ USID}$.

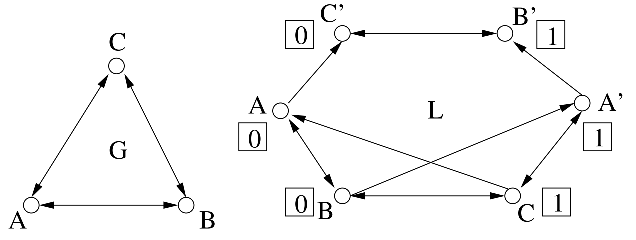

Figure 3: Graph G and System L.

Formally, system L is a synchronous system with a well defined output distribution for any particular input assignment. We show that for a particular input assignment, no such well defined behavior is possible. Further, no player in L knows the complete system. Each player in aware of only his immediate neighbors. Let E_1 and E_2 be two parallel executions of $\Pi_{2,\ USID}$ over G. Let, α_1 be a scenario in E_1 where A is the General starting with input 0 and adversary $\mathcal{A}$ corrupts C. Let α_2 be another scenario in E_1 where A is the General, $\mathcal{A}$ corrupts A and makes it to behave with B as if it started with input 0 & behave with C as if it started with input 1. In scenario α_3 in E_1, A is the General starting with input 1 and $\mathcal{A}$ corrupts B. Similarly, let α_4 be a scenario in E_2 where $\mathcal{A}$ corrupts A. Further, let α be an execution of L where each player starts with input value as shown in Figure 3. All the players in α are honest and follow the designated protocol correctly.

We claim that in E_1, $\mathcal{A}$ can ensure that whatever *view* (informally view of a player refers to all the messages the player ever gets to see during the entire protocol execution. Formal definition of view is given in [**?**]) A, B get in α, $\mathcal{A}$ can generate the same view for A, B in α_1 i.e. both A and B cannot ever differentiate between α_1 and α (dubbed $\alpha_1 \overset{A}{\sim} \alpha$ and $\alpha_1 \overset{B}{\sim} \alpha$). From the definition of ABG [Definition 1], in α_1, both A, B should decide on value 0. Since view of A, B is same in α_1 and α, both A, B in α will also decide on value 0. Similarly, both A, C in α_3 should decide on value 1. $\mathcal{A}$ can ensure that $\alpha_3 \overset{C}{\sim} \alpha$ and $\alpha_3 \overset{A'}{\sim} \alpha$. Thus both A', C in α will decide on value 1. Similarly B, C in α_2 should agree on same value, then so should B, C in α. But B,C have already decided upon values 0 and 1 respectively in α. This implies L must exhibit contradictory behavior. This contradicts our assumption of a protocol $\Pi_{2,\ USID}$ over G tolerating 2-adversary.

To complete the proof sketch, we show as to how $\mathcal{A}$ can ensure that $\alpha_1 \overset{A}{\sim} \alpha$ and $\alpha_1 \overset{B}{\sim} \alpha$. Consider a run Γ of L which is exactly same as α except that in Γ A' starts with input value 0. Since in α, no message from B' or C' can ever reach any of A,B,C or A', $\mathcal{A}$ can ensure that A and B get same messages in Γ and α_1 (All $\mathcal{A}$ has to do is to make C start with value 1 and follow the designated protocol). Now in α, all messages received by A and B respectively are same as those in Γ except those messages that have been processed by A' at least once(since A' starts with input value 0 in Γ and input value 1 in α). If in α_1, $\mathcal{A}$ can simulate this difference between α and Γ, we can say that $\mathcal{A}$ can make view of A and B same in α and α_1. We now claim that for any round i, $i \geq 1$, it is always possible for $\mathcal{A}$ to do so. Note that owing to the typical construction of S, in α A' can send a message to A (and B) only via C. This ensures that in α, any message from A' can reach A (and B) only after it has been processed by C. Now in α_1, C is faulty, so if $\mathcal{A}$ can generate messages in α_1 similar to messages sent by C to A (and B) in α, it can make the views same. In α_1, A is honest and starts with input 0. However, since A is corrupt in α_4, $\mathcal{A}$ can read the private key used by A for authenticating his messages. This essentially means $\mathcal{A}$ can forge messages on behalf of A. Specifically, in round i of α_1, $\mathcal{A}$ sends to A and B what an honest C would have sent to A and B in α_1 if A would have started with input value 1. Note that messages sent by A in round i of α_1 depends on the internal state of A till round $i-1$ of α_1 and the code executed by A. The internal state consists of player A's input value, authentication key, session identifier used for this execution (E_1) and messages received by A till round $i-1$. Note that since A is corrupt in α_4, $\mathcal{A}$ can always pull the all the relevant information regarding internal state of A in α_1 via A in α_4 [3]. Similarly one can show that $\mathcal{A}$ can always ensure $\alpha_2 \overset{B}{\sim} \alpha$ and $\alpha_2 \overset{C}{\sim} \alpha$, and $\alpha_3 \overset{C}{\sim} \alpha$ and $\alpha_3 \overset{A'}{\sim} \alpha$ ■

We now formally define π' and prove that π' exists if $\Pi_{2,\ USID}$ exists.

Definition 3 **(π')** *For all players $a, b \in \mathbb{P}$, any statement in $\Pi_{2,\ USID}$ of the kind "b sends message m to a" is replaced by "b multicasts message m to all instances of a(i.e. a,a') [4] which are connected by a directed edge from b to a" in π'. Similarly any statement of the kind "c sends message m to a" is replaced by "c multicasts message m to all instances of a. Rest all statements in π' are same as those in $\Pi_{2,\ USID}$.*

Lemma 2 *If $\Pi_{2,\ USID}$ exists, then π' exists.*

Proof: Implied from Definition 3. ■

Formally, one can prove the following lemma. Here $view_X^{\phi}$ represents view of player X during entire execution ϕ. Detailed proofs are given in [?].

Lemma 3 *Adversary $\mathcal{A}$ can ensure the following:*

$view_A^{\alpha} \sim view_A^{\alpha_1}$ *and* $view_B^{\alpha} \sim view_B^{\alpha_1}$.

$view_B^{\alpha} \sim view_B^{\alpha_2}$ *and* $view_C^{\alpha} \sim view_C^{\alpha_2}$.

$view_{A'}^{\alpha} \sim view_A^{\alpha_3}$ *and* $view_C^{\alpha} \sim view_C^{\alpha_3}$.

As an interesting observation, it appears that the proof of Lemma 3 requires *directed* system, unlike undirected systems used in extant literature [16, 29].

5 Characterization of ABG under Parallel Composition

We now give the necessary and sufficient conditions for existence of $\Pi_{k,\ USID}$ over any completely connected synchronous network. We first show impossibility of $\Pi_{2,\ USID}$ over a complete graph H (Figure 4) of four nodes $\mathbb{P} = \{A, B, C, D\}$ tolerating adversary basis $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$.

[3]using our earlier visualization of a player running concurrent executions as a processor running concurrent threads, $\mathcal{A}$ can always ask A to execute a code in thread E_2 which can pull information out of another thread E_1 being run in same processor.

[4]a and a' are independent copies of the player a with same authentication key.

Here $((x_1 \dots x_i)(y_1 \dots y_j))$ represents a single element of adversary basis such that adversary can corrupt all $x_1 \dots x_i$ in one execution and corrupt all $y_1 \dots y_j$ in the second concurrent execution. The proof technique is similar to one used for proving impossibility of 2-out-of-3 (Theorem 1) in section 4.

Theorem 4 *There does not exist any protocol $\Pi_{2,\ USID}$ over a complete graph H of four nodes, tolerating adversary basis $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$.*

Proof sketch: We assume there exists a protocol $\Pi_{2,\ USID}$ tolerating adversary basis $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$ over a complete graph H (Figure 4). We show that there exist two parallel executions of $\Pi_{2,\ USID}$, where the real process adversary $\mathcal{A}$ (characterised by $\bar{\mathbb{A}}$) can ensure that honest players in one of the executions do not have consistent output. In the corresponding ideal execution honest players are guaranteed to have a consistent output. Thus there *does not* exist any ideal process adversary $\mathcal{S}$ which can ensure that the output distributions are indistinguishable, thus violating Definition 1.

Similar to proof of Theorem 1, from $\Pi_{2,\ USID}$ we create a protocol η in such a way that if $\Pi_{2,\ USID}$ exists then so does η. Using two copies of η, we construct a system M (as shown in Figure 4), and show that M must exhibit contradictory behavior. This contradicts our assumption about existence of $\Pi_{2,\ USID}$.

We do not know what system M solves. All we know is that M is a synchronous system with a well defined output distribution for any particular input assignment. We show that for a particular input assignment, no such well defined behavior is possible. Further no player in L knows the complete system. Each player in aware of only his immediate neighbors. Let E_1, E_2 be two parallel executions of $\Pi_{2,\ USID}$ over H. Let β_1, β_2 and β_3 be three distinct scenarios in E_1. In β_1, B is the General starting with input 0 and A,D are corrupt. Similarly, in β_2 B is the General. Adversary corrupts B and makes him behave with C as if it started with input 0 & behaves with A, D as if it started with input 1. In β_3, D is the General with input value 1 and adversary corrupts C. In E_2, β_4 is a scenario where $\mathcal{A}$ corrupts B and in β_5, $\mathcal{A}$ corrupts A. Further, let β be an execution of M where each player starts with input value as shown in Figure 4. All the players in β are honest and follow the designated protocol correctly.

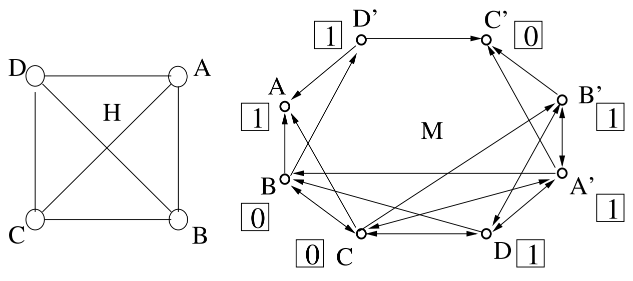

Figure 4: Graph H and System M.

Similar to proof of Theorem 1, one can show that whatever view B, C get in β, adversary can ensure that B, C get the same in β_1. As per Definition 1, in β_1, both B, C should decide on value 0. Then so should B, C in β. Similarly, whatever view A', B' and D get in β, adversary can ensure the same for A, B and D respectively in β_3. Since A, B and D in β_3 decide upon 1, then so should A', B' and D in β. Whatever view C, D and A' get in β, adversary can ensure the same for C, D and A respectively in β_2. As per Definition 1, in β_2, all C, D and A are required to output same value. Then so should be for C, D and A' in β. But in β, C and D, A' have already decided on 0 and 1 respectively. Thus M exhibits a contradictory behavior. ∎

Similar to section 4, one can prove the following Lemma.

Lemma 5 *Adversary can ensure the following:*

$view_B^{\beta} \sim view_B^{\beta_1}$ *and* $view_C^{\beta} \sim view_C^{\beta_1}$.

$view_C^{\beta} \sim view_C^{\beta_2}$, $view_D^{\beta} \sim view_D^{\beta_2}$, $view_{A'}^{\beta} \sim view_A^{\beta_2}$.

$view^{\beta}_{A'} \sim view^{\beta_3}_{A}$, $view^{\beta}_{B'} \sim view^{\beta_3}_{B}$, $view^{\beta}_{D} \sim view^{\beta_3}_{D}$.

We now give the main theorem of this paper.

Theorem 6 (Main Theorem) *There exists a protocol $\Pi_{k,\ USID}$ tolerating t-adversary over a completely connected graph of n nodes if and only if $n \geq 2t$.*

Proof: `Necessity:` We first prove impossibility of any protocol ($\eta_{2,USID}$) using USID solving ABG that composes in parallel even twice over a complete graph of n nodes for $n \leq 2t_1 + min(t_1, t_2)$, $t_2 > 0$. Here t_1,t_2 refer to the number of players the t-adversary corrupts in two parallel executions E_1 and E_2 respectively such that t_1+$t_2 \leq t$ (dubbed as (t_1,t_2)-adversary). Then using t_1=$t - 1$ and t_2=1 in $n \leq 2t_1 + min(t_1, t_2)$ one gets the impossibility for $n < 2t$.

To prove the impossibility of $\eta_{2,USID}$, we start by assuming that there exists a protocol $\eta_{2,USID}$ over a complete graph of n nodes tolerating (t_1,t_2)-adversary when $n \leq 2t_1 + min(t_1, t_2)$, $t_2 > 0$. Using $\eta_{2,USID}$ we construct a protocol $\Pi_{2,\ USID}$ over a complete graph of four nodes $\{A, B, C, D\}$, tolerating $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. We then show that if $\eta_{2,USID}$ satisfies definition 1, then so does $\Pi_{2,\ USID}$. But this contradicts Theorem 4. Thus our assumption that there exists a solution $\eta_{2,USID}$ for $n \leq 2t_1 + min(t_1, t_2)$ is wrong.

We now show as to how $\eta_{2,USID}$ can be transformed into a solution $\Pi_{2,\ USID}$ for four players completely connected, tolerating $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. Divide n players into four sets: I_A, I_B, I_C, I_D, such that their respective sizes are $min(t_1, t_2), min(t_1, t_2), t_1, (t_1 - min(t_1, t_2))$. Let E_1 and E_2 be the two parallel executions of $\eta_{2,USID}$. Adversary $\mathcal{A}$ can corrupt any of the following sets $I_A, I_B, I_C, I_D, (I_A \cup I_D), (I_B \cup I_D)$ in E_1 and any of the sets I_A, I_B, I_D in E_2. Let the corresponding two parallel executions of $\Pi_{2,\ USID}$ be E'_1 and E'_2. Each of the four players A,B,C and D in execution E'_i simulates all the players in I_A, I_B, I_C, I_D respectively in execution E_i. Player i in E'_i simulates players in I_i in E_i as follows: player i keeps track of the states of all the players in I_i. Player i assigns its input value to every member of I_i, and simulates the steps of all the players in I_i as well as the messages sent and received between pairs of players in I_i. Messages from players in I_i to players in I_j are simulated by sending same messages from player i to player j. If any player in I_i terminates then so does player i. If any player in I_i decides on a value v, then so does player i.

We now show that if $\eta_{2,USID}$ satisfies definition 1 when $n \leq 2t_1 + min(t_1, t_2)$, $t_2 > 0$, then so does $\Pi_{2,\ USID}$ tolerating $\bar{\mathbb{A}} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. Consider two honest players i and j ($i \neq j$) in execution E'_i. Each of them simulates atleast one player in I_i and I_j in execution E_i. Since both i and j are honest in E'_i, then so are all the players in I_i and I_j in execution E_i. If the General Gen is corrupt in E'_i, then so is the General in E_i. If players in I_i, I_j in execution E_i decide on value u, then so does players i, j in E'_i. If the General is honest in E'_i and starts with a value v, then in E_i too the General is honest and starts with a value v. Then as per definition 1 all the players in I_i, I_j in execution E_i decide on value v, then so should players i, j in E'_i. This implies $\Pi_{2,\ USID}$ satisfies definition 1 and tolerates $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. But from Theorem 4, we know there does not exist any such $\Pi_{2,\ USID}$. This contradicts our assumption of $\eta_{2,USID}$. This completes the necessity proof.

`Sufficiency:` For sufficiency, we give a protocol for ABG using USIDs for $n \geq 2t$ and prove its correctness in stand alone setting. The protocol is given in section 5.1. We now prove that the proposed protocol composes for any number of parallel executions. The proof technique is essentially similar to one developed by Lindell *et al.* [29, 30] wherein the security of the protocol under composition is reduced to security of the protocol in stand alone setting. In order to accommodate the fact that under parallel composition adversary can still forge signatures of a fraction of honest players, we tweak the stand alone settings as follows: Consider the adversary as (t_b, t_p)-adversary where by adversary can corrupt up to any t_b players actively and another t_p players passively such that adversary can forge signatures of all $t_b + t_p$ players. We further require all the passively corrupt players to be always part of the agreement with the condition that if the *Gen* in not Byzantine corrupt and starts with input value v, then all honest and passively corrupt players should decide on v. For this problem, Gupta *et al.* [24] prove the tight bound of $n > 2t_b + min(t_b, t_p)$. Putting t_b=$t-1$, t_p=1, one gets $n \geq 2t$ as necessary and sufficient for this problem. Let $\pi(1) \ldots \pi(l)$ be l parallel executions of our **EIG** protocol. We now prove that if there exists an adversary that can attack and succeed in some execution $\pi(i)$, $i \in (1, l)$, then we can construct an adversary $\mathcal{A}'$ that is bound to succeed against stand alone execution of the protocol considered by Gupta *et al.*

Formally, let $\pi(id_1) \ldots \pi(id_l)$ be l parallel executions of **EIG** protocol where execution $\pi(id_i)$ uses session identifier id_i. Let there exists an adversary $\mathcal{A}$ that succeeds in some execution $\pi(i)$. $i \in (1, l)$. Players use signature scheme $((Gen,S_{id},V_{id}),S_{\neg id})$ developed by Lindell *et al.* [29, 30]. Using $\mathcal{A}$ we construct adversary $\mathcal{A}'$ that is bound to succeed against stand alone execution $\Pi(id)$ of the protocol considered by Gupta *et al.* Let players in $\pi(id_i)$ be partitioned into 3 parts I_b, I_p and I_h where I_b are those which are byzantine faulty in $\pi(id_i)$, I_p are those which are honest in $\pi(id_i)$ but corrupt in execution $\pi(id_k)$, $k \neq i$, $k \in \{1 \ldots l\}$ and I_h are those which are honest in $\pi(id_i)$ as well as all other executions $\pi(id_k)$, $k \neq i$, $k \in \{1 \ldots l\}$. Further let X_b be Byzantine faulty, X_p be passively corrupt and X_h be honest players in $\Pi(id)$. Let all the players in X_i simulate all the players in I_i as follows: each player in X_i keeps track of the states of all the players in I_i. Player i assigns its input value to every member of I_i, and simulates the steps of all the players in I_i as well as the messages sent and received between pairs of players in I_i. Messages from players in I_i to players in I_j are simulated by sending same messages from X_i to every player in X_j. If any player in I_i terminates, then so does all the players in X_i. If any player in I_i decides on value v, then so does all the players in X_i.

$\mathcal{A}'$ internally incorporates $\mathcal{A}$ and attacks $\Pi(id)$ as follows: $\mathcal{A}'$ randomly selects an execution $i \in \{1 \ldots l\}$ and sets id=id_i. Then $\mathcal{A}'$ invokes $\mathcal{A}$ and emulates the concurrent executions of $\pi(id_1) \ldots \pi(id_l)$ for $\mathcal{A}$. $\mathcal{A}'$ does this by playing the roles of the honest players in all but the execution $\pi(id_i)$. In $\pi(id_i)$, $\mathcal{A}'$ externally interacts with the honest players and passes messages between them and $\mathcal{A}$. Since $\mathcal{A}'$ is given access to the signing oracles $S_{\neg id}(sk_1, .), \ldots, S_{\neg id}(sk_n, .)$, it can generate signature on behalf of honest players in all execution $\pi(id_j)$, $j \neq i$. The proof hinges on the fact that in $\pi(id_i)$, $\mathcal{A}$ can forge signature on behalf of only those honest players which belong to set I_p. Note that in $\Pi(id)$, players corresponding to I_p are those in set X_p. Since players in X_p are passively corrupt, $\mathcal{A}'$ can forge signature on behalf on any player belonging to X_p. Thus whatever messages $\mathcal{A}$ can forge in $\pi(id_i)$, $\mathcal{A}'$ can forge the same in $\Pi(id)$. Therefore, the emulation by $\mathcal{A}'$of the concurrent executions for $\mathcal{A}$ is perfect. Thus if $\mathcal{A}$ succeeds in breaking $\pi(id_i)$, then $\mathcal{A}'$ should also succeed in breaking $\Pi(id)$. But from the results of Gupta *et al.* [24], we know that there cannot exist any such $\mathcal{A}'$. This contradicts our assumption about existence of $\mathcal{A}$. ■

5.1 Protocol for $n \geq 2t$

The proposed protocol is obtained by a sequence of transformations on *EIG* [2]. A detailed description of the construction of *EIG* tree is available in [31, page 108]. The General sends his input to every player. Each player starts with this as input value and exchanges messages with others as per *EIGStop* protocol in [31, page 110] for $t + 1$ rounds. The reason for giving a EIG based protocol is its ease of understanding. Using well known techniques given in [2], our **EIG** protocol can be converted into an efficient protocol.

Definition 4 (Prune(*EIG*)) *: This method takes an* EIG *tree as an input and returns it with subtrees say subtree$_j{}^i$ (subtree$_j{}^i$ refers to a subtree in i's* EIG *tree such that the subtree is rooted at node whose's label is j) of i's* EIG *tree deleted. The criteria for selecting subtree$_j{}^i$ is as follows: for each subtree subtree$_j{}^i$, where label $j \in \mathbb{P}$, a set W_j is constructed which contains all distinct values that ever appear in subtree$_j{}^i$. If $|W_j| > 1$, subtree$_j{}^i$ is deleted else not.*

At the end of $t+1$ rounds of *EIGStop* protocol, we invoke **Prune**(*EIG*). Player i applies the following decision rule. Namely, Player i takes a majority of the values at the first level [5] of its *EIG* tree (note that he does not need to take a majority over the entire *EIG* tree). If a majority exists, player i decides on that value; otherwise, i decides on a *default value*, v_0.

Informally the correctness of protocol in standalone model is evident from that fact that $\mathcal{A}$ can forge messages only on behalf of at most t players. And subtrees of all such faulty players in any honest player's EIG tree will be deleted. This is because the flood set protocol ensures that if one honest player ever gets two different values from any player, then so does every other honest player. We only state our lemmas.

Lemma 7 *The* $subtree_j{}^i$*, i and j are honest players, never gets deleted by* **Prune**(EIG) *operation.*

Lemma 8 *After $t+1$ rounds, if a* $subtree_j{}^i$ *has more than one value then* $\forall$ *k,* $subtree_j{}^k$ *also has more than one value, there by ensuring that all* $\forall$ *k,* $subtree_j{}^k$ *are deleted (i, j, k are not necessarily distinct), where i, k are honest.*

Lemma 9 $subtree_j{}^i$ *and* $subtree_j{}^k$ *in the* EIG *trees of any two honest players i, k will have same values after the subjecting the tree to* **Prune**(EIG).

We now show that our protocol meets definition 2.

Lemma 10 ***EIG*** *solves ABG as per definition 2.*

Proof sketch: The decision rule ensures that if the *Gen* is honest and starts with value v, then all honest players at the end of the protocol output value v. The ideal process adversary $\mathcal{S}$ cannot not corrupt the *Gen*. In the corresponding ideal process execution, the *Gen* starts with value v and thus every honest players decides on v. If the real world adversary $\mathcal{A}$ corrupts the *Gen*, as per decision rule, all honest players decide on v_0. In the corresponding ideal process execution, $\mathcal{S}$ corrupts the *Gen* and on behalf of *Gen* sends value v_0 to TTP. Thus all honest players in ideal execution will output v_0. $\mathcal{S}$ always ensures that faulty players in ideal process execution always output same as what they output in the protocol execution. This ensures that the ensembles $IDEAL_{TTP,\mathcal{S}}$ and $REAL_{Prune(EIG),\mathcal{A}}$ are computationally indistinguishable. ■

6 Conclusion

Unique session identifiers aid in improving the fault-tolerance of ABG protocols (that compose in parallel) from $n > 3t$ to $n \geq 2t$. Note that stand-alone ABG is possible for $n > t$. Thus surprisingly, USID's may not always achieve their goal of truly *separating* the protocol's execution from its environment. However, for most functionalities, USID's indeed achieve their goal, as is obvious from Canetti's universal composition theorem [6]. The anomaly with respect to ABG, as pointed out in Section 3, is that the worst-case adversary (with respect to a given execution) is not the one that corrupts players at full-throttle across all protocols running concurrently in the network. Therefore, there may be several other problems apart from ABG which could potentially hinder with the power and role of USID's. It is an intriguing open question to characterize the set of all such problems.

[5] all nodes with labels l such that $l \in \mathbb{P}$.

7 On Contradiction with the Literature

We now elaborate on the shortcoming in the proof for sufficiency of $n > t$ for protocols with USIDs for parallel composition of ABG [29, 30]. For the benefit of the reader, a brief overview of the proof is given in Appendix A. We claim that the proof implicitly assumes that the adversary cannot corrupt different players in different parallel executions. We now formally show that if above mentioned assumption does not hold, the proof breaks down.

Formally, the proof assumes that under concurrent executions, for a particular execution $\pi(id_k)$, $\mathcal{A}$ cannot ever forge signature of any honest player in $\pi(id_k)$. This is because $S_{\neg id_k}(sk,m) = \perp$ in case the prefix of message $m = id_k$. However, if the adversary chooses to corrupt different players in different executions, then for all honest players in $\pi(id_k)$, $S_{\neg id_k}(sk,m) = \perp$ *need not necessarily be true*. This is because $\mathcal{A}$ may choose to corrupt a particular player (who is honest in $\pi(id_k)$) in some other execution, forge messages on behalf of him and use the same in $\pi(id_k)$. Specifically, let p_1 be an honest player in execution $\pi(id_k)$. $\mathcal{A}$ corrupts p_1 in some other execution say $\pi(id_l)$. In order to use a forge message say m' on behalf of p_1 in some round i of $\pi(id_k)$, $\mathcal{A}$ needs to construct m' on behalf of p_1 with session identifier of $\pi(id_k)$. For this $\mathcal{A}$ needs the signature key and the internal state of p_1 just prior round i in $\pi(id_k)$ (Internal state of a player at an round includes his input value, signature key, USID, and view of the player till this round). Since p_1 is an honest player in $\pi(id_k)$, $\mathcal{A}$ cannot do the forgery in execution $\pi(id_k)$. However, in $\pi(id_l)$ p_1 is corrupt. Via p_1 in execution $\pi(id_l)$, $\mathcal{A}$ can pull signing key and internal state of p_1 just before round i in $\pi(id_k)$. Using this $\mathcal{A}$ can construct relevant message m' such that m' uses the session identifier of $\pi(id_k)$. Since m' uses same session identifier as used for $\pi(id_k)$, in $\pi(id_k)$ m' will not be rejected by any protocol which filters messages based on incorrect session identifiers. Note that since P_A is an honest player in $\pi(id_k)$, this amounts to forgery.

8 Related Work

Byzantine Generals Problem (BGP) was first introduced by Pease *et al.* [27]. It is well known that BGP over completely connected synchronous network is possible if and only if $n > 3t$ [32, 27]. Later on the problem was studied in many different settings [17, 14, 13, 19, 18, 1, 21, 20, 34] to name a few, giving both possibility (protocols) and impossibility results. An important variant of BGP is *authenticated Byzantine Generals* (ABG) introduced by [27, 32], where the players are supplemented with 'magical' powers (say a Public Key Infrastructure(PKI) and digital signatures) to authenticate themselves and their messages. Pease *et al.* proved that for such a model, tolerability against a t-adversary can be amazingly increased to $n > t$ which is a huge improvement over $n > 3t$. Subsequent work in this subject includes [12, 3, 5, 36, 4, 26, 23, 35, 30].

Some of the first work on composition of protocols was on the problem of zero-knowledge and concurrent zero-knowledge [11, 15, 22]. Canetti [6] introduced the notion of *Universal Composability* to study the implications on security of protocols when run in arbitrary any unknown protocols. Some of the subsequent papers in this line are [28, 10, 9, 33, 8, 7]. In continuation, Lindell *et al.* [29, 30] introduced the problem of ABG under parallel composition. They proved for $n < 3t$ there *does not* exist any protocol solving ABG that composes in parallel even twice. They further prove that protocols for ABG over a completely connected synchronous network of n players, tolerating t-adversary, compose in parallel (for any number of executions) if and only if $n > 3t$. In the same work, they show that if one assumes additional facility of *unique session identifiers*, fault tolerance for ABG under parallel composition can be restored back to $n > t$.

References

[1] Bernd Altmann, Matthias Fitzi, and Ueli M. Maurer. Byzantine agreement secure against general adversaries in the dual failure model. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 123–137, London, UK, 1999. Springer-Verlag.

[2] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: changing algorithms on the fly to expedite byzantine agreement. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 42–51, New York, NY, USA, 1987. ACM Press.

[3] Malte Borcherding. On the number of authenticated rounds in byzantine agreement. In *WDAG '95: Proceedings of the 9th International Workshop on Distributed Algorithms*, pages 230–241, London, UK, 1995. Springer-Verlag.

[4] Malte Borcherding. Levels of authentication in distributed agreement. In *WDAG '96: Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 40–55, London, UK, 1996. Springer-Verlag.

[5] Malte Borcherding. Partially authenticated algorithms for byzantine agreement. In *ISCA: Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems*, pages 8–11, 1996.

[6] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001. Full version available at `http://eprint.iacr.org/2000/067`.

[7] R. Canetti and M. Fischlin. Universally Composable Commitments. In *Proceedings of Advances in Cryptology CRYPTO '01*, volume 2139 of *Lecture Notes in Computer Science*, pages 19 – 40. Springer-Verlag, 2001.

[8] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Proceedings of Advances in Cryptology - EUROCRYPT '02*, volume 2332 of *Lecture Notes in Computer Science (LNCS)*, pages 337–351. Springer-Verlag, 2002.

[9] R. Canetti and T. Rabin. Universal Composition with Joint State. In *Proceedings of Advances in Cryptology - CRYPTO '03*, volume 2729 of *Lecture Notes in Computer Science (LNCS)*, pages 265–281. Springer-Verlag, 2003.

[10] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[11] Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires Ω (logn) rounds. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 570–579, New York, NY, USA, 2001. ACM.

[12] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[13] Danny Dolev. The byzantine generals strike again. Technical report, Stanford, CA, USA, 1981.

[14] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987.

[15] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[16] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. In *PODC '85: Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 59–70, New York, NY, USA, 1985. ACM.

[17] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[18] Matthias Fitzi and Ueli M. Maurer. Efficient byzantine agreement secure against general adversaries. In *International Symposium on Distributed Computing*, pages 134–148, 1998.

[19] Mattias Fitzi and Ueli Maurer. From partial consistency to global broadcast. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 494–503, New York, NY, USA, 2000. ACM.

[20] J. A. Garay. Reaching (and Maintaining) Agreement in the Presence of Mobile Faults. In *Proceedings of the 8th International Workshop on Distributed Algorithms – WDAG '94*, volume 857 of *Lecture Notes in Computer Science (LNCS)*, pages 253–264, 1994.

[21] Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In *WDAG '92: Proceedings of the 6th International Workshop on Distributed Algorithms*, pages 153–165, London, UK, 1992. Springer-Verlag.

[22] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[23] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults, 1995.

[24] Anuj Gupta, Prasant Gopal, Piyush Bansal, and Kannan Srinathan. Authenticated Byzantine Generals Strike Again. Technical report, Center for Security, Theory and Algorithmic Research (CSTAR), International Institute of Information Technology, Hyderabad, India, 2008. A complete version is available at http://research.iiit.ac.in/~anujgupta/BA_with_authentication_web_version.pdf.

[25] Anuj Gupta, Prasant Gopal, Piyush Bansal, and Kannan Srinathan. Composition of authenticated Byzantine Generals Revisited. Technical report, Center for Security, Theory and Algorithmic Research (CSTAR), International Institute of Information Technology, Hyderabad, India, 2008. A complete version is available at http://research.iiit.ac.in/~anujgupta/Composition_of_ABG_report.pdf.

[26] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. 2007.

[27] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[28] Y. Lindell. *Composition of Secure Multi-Party Protocols: A Comprehensive Study*, volume 2815 of *Lecture Notes in Computer Science (LNCS)*. Springer–Verlag, 2003.

[29] Y. Lindell, A. Lysysanskaya, and T. Rabin. On the Composition of Authenticated Byzantine Agreement. In *Proceedings of the 34th Symposium on Theory of Computing (STOC)*, pages 514–523. ACM Press, 2002.

[30] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *J. ACM*, 53(6):881–917, 2006.

[31] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, San Mateo, CA, USA, 1996.

[32] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[33] M. Prabhakaran and A. Sahai. New Notions of Security: Achieving Universal Composability without Trusted Setup. In *Proceedings of the 36th Symposium on Theory of Computing (STOC)*, pages 242–251. ACM Press, June 13–15 2004.

[34] M. O. Rabin. Randomized byzantine generals. In *Proc. of the 24th Annu. IEEE Symp. on Foundations of Computer Science*, pages 403–409, 1983.

[35] Ulrich Schmid and Bettina Weiss. Synchronous byzantine agreement under hybrid process and link failures. Research Report 1/2004, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2004.

[36] T. K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

A Appendix

We present an overview of the proof for sufficiency of $n > t$ for parallel composition of protocols for ABG in USID model. [29, 30]. The proof essentially reduces the security of protocols for ABG with USIDs for any number of parallel compositions to the security of a stand alone execution of the protocol. They define a signature scheme as (Gen,S,V) where S,V are are algorithms for signing and verification of any message. Gen is used to generate signature and verification keys for a particular player (say P_k) and defined as a function: $(1)^n \rightarrow (vk, sk)$. A signature scheme is said to be a valid one if honestly generated signatures are almost always accepted. Formally, with non negligible probability, for every message m, V(vk,m,S(sk,m)) = 1, where $(vk, sk) \leftarrow (1)^n$. They model the valid signatures that adversary $\mathcal{A}$ can obtain in a real attack via a signing oracle $S(sk, \cdot)$. $\mathcal{A}$ is defined to succeed in generating a forged message m^* if $\mathcal{A}$ given vk, access to oracle $S(sk, \cdot)$ can generate a pair (m^*, σ^*) such that if Q_m is the set of oracle queries made by $\mathcal{A}$ then V(vk,m^*,σ^*) = 1 holds true if $m^* \notin Q_m$. A signature scheme is said to be existentially secure against chosen-message attack if $\mathcal{A}$ cannot succeed in forging a signature with greater than non-negligible probability. They further model any information gained by $\mathcal{A}$ from any query with another oracle Aux(sk,.). However, this oracle cannot generate any valid signature but provides any other auxiliary information about the query. They assume some scheme say (Gen,S,V) to be secure against chosen-message attack and show how to construct a secure scheme (Gen,S_{id},V_{id}) from it where $S_{id}(sk,m) = S(sk,id \circ m)$ and $V_{id}(vk,m,\sigma) = V(vk,id \circ m,\sigma)$. For the new scheme they define the oracle Aux(sk,$\cdot$) = $S_{\neg id}(sk,m)$ where $S_{\neg id}(sk,m) = S(sk,m)$ if the prefix of m is not id else $S_{\neg id}(sk,m) = \perp$. Further, they assume π to be a secure protocol for ABG using signature scheme (Gen,S,V). They define modified protocol $\pi(id)$ to be exactly same as π except that it uses signature scheme (Gen,S_{id},V_{id}) as defined above. They further prove as to why ((Gen,S_{id},V_{id}), $S_{\neg id}$) is secure against chosen-message attack. Intuition behind the proof is the fact that if the prefix of $m \neq id$, then $S_{\neg id}(sk,m) = S(sk,m)$ which is of no help to the adversary as any successful forgery must be prefixed with id and all oracle queries to $S_{\neg id}$ must be prefixed with $id' \neq id$. Formally they show how an adversary $\mathcal{A}'$ for a single execution of $\pi(id)$ can simulate an adversary $\mathcal{A}$ for concurrent executions $\pi(id_1) \ldots \pi(id_l)$, thus reducing the security of concurrent executions to security of stand alone execution. If $\mathcal{A}$ attacks concurrent executions and succeeds in breaking in some execution say $\pi(id_i)$, then $\mathcal{A}'$ can internally incorporate $\mathcal{A}$ and succeed in breaking single execution $\pi(id_i)$. $\mathcal{A}'$ randomly selects an execution $i \in \{1, \ldots l\}$ and sets id=id_i. $\mathcal{A}'$ invokes $\mathcal{A}$ and emulates concurrent executions $\pi(id_1) \ldots \pi(id_l)$ for $\mathcal{A}$. $\mathcal{A}'$ does so by playing roles of honest players in all but the i^{th} execution $\pi(id_i)$. In $\pi(id_i)$, $\mathcal{A}'$ externally interacts with the honest players and passes messages between them and $\mathcal{A}$. Since $\mathcal{A}'$ has access to signing oracles $S_{\neg id}(sk_1), \ldots S_{\neg id}(sk_n)$, $\mathcal{A}'$ can generate messages on behalf of honest players in all executions $\pi(id_j)$ for $j \neq i$. This implies that $\mathcal{A}'$ can perfectly simulate $\mathcal{A}$, thus $\mathcal{A}'$ should be able to break security of stand alone execution of $\pi(id_i)$.