Authenticated Byzantine Generals

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science (by Research)
in
Computer Science

by

Anuj Gupta
200607001
anujgupta@research.iiit.ac.in

International Institute of Information Technology
Hyderabad, India
July 2009

Thesis Certificate

This is to certify that the thesis entitled **"Authenticated Byzantine Generals"** submitted by **Anuj Gupta** to the International Institute of Information Technology, Hyderabad, for the award of the Degree of **Master of Science (by Research)** is a record of bona-fide research work carried out by him under my supervision and guidance. The contents of this thesis have not been submitted to any other university or institute for the award of any degree or diploma.

Hyderabad 500032 Research Guide

Date Dr. Kannan Srinathan

To Parameshwara and my Parents

"*I do not know what I may appear to the world; but to myself I seem to have been only like a boy playing on the sea-shore, and diverting myself in now and then finding a smoother pebble or a prettier shell than ordinary, whilst the great ocean of truth lay all undiscovered before me.*"

- Isaac Newton

Acknowledgements

First n foremost, I would like to thank Dr. Kannan Srinathan who has had a tremendous influence on me. On top of being an expert, he has been a dear friend, a profound philosopher and a devoted guide, far beyond ordinary. He introduced me to research and closely guided my first steps. He has this special property of always searching for the crux of any matter. This, together with his sharpness, energy and amazing sense of humour makes him a remarkable person and pleasure to work with. His feedback and encouragement greatly helped me to keep my spirits up.

He always encouraged me to develop a broader perspective towards research and life in general. We would spent hours on discussing over a wide variety topics ranging from cricket, education to the realms of life, human existence and what not ! Today, as I look back, I find these discussion very close to my heart and by far the most cherished part of my journey. These interactions introduced me to a newer perspective of looking at things around me and have greatly influenced my understanding of life in general.

I also would like to acknowledge the members of my graduate review committee namely, Prof. C. V. Jawahar, Dr. Navin Goel, and Dr. Suresh Purini for their encouragement and invaluable suggestions.

I am also thankful to all the faculty members of CSTAR - Prof. V. Ch. Venkaiah, Dr Kannan Srinathan, Dr. Kishore Kothapalli and Dr. Bruhadeshwar Bezawada for providing a wonderful research center. I also take this opportunity to thank IIIT-Hyderabad for giving me an opportunity to see the world of research so closely.

My research mates Prasant Gopal, Piyush Bansal, Pranav Kumar Vasishta, Sandeep Hans and Sarat Addepalli deserve a special acknowledgement. Without these people this thesis would not have been possible. I would also like to thank my seniors Ritesh Kumar Tiwari, Ananda Swarup Das and G. Bhavani Shankar for their guidance. My lab-mates were very kind and helpful. They include - Pankaj Kohli, V. Sai Satyanarayana, Abhinav Mehta, G. Uma Devi, Rohit Ashok Khot, Charaka Vinayak Kumar, Neeraj Kumar, Kaushik Gampa, Romanch Agrawal and M. Poornima.

This acknowledgment would be incomplete without mention of my batch mates who made my journey memorable. This includes - T.V. Kalyan, Mahesh Mohan, PRASHANTH PAI B, S.S.Keerthi Ram, Md. Abid Hussain, Amit Khandelwal, Siddhartha Reddem, Amol Panchal, Sandeep Thorat, Nayan Mujadiya, K. Rahul, Chandan Kumar, Subhash S, Jyotika Bahuguna. I am also thankful to my friends Mayur, Rakhi, Amit, Akshat, Shreyansh, Sunder, Shalini and Debashish for their support and encouragement.

Above all I would like to express my gratitude towards my parents who have had, are having and will continue to have a tremendous influence on my development. I am deeply thankful to Bhaiya for his eternal love and support.

Abstract

The problem of simulating a broadcast channel over a point to point unreliable network is a fundamental problem in theory of distributed computing. The problem is popularly known as "Byzantine Generals Problem" (BGP). It involves reliable communication of a message m from a designated sender (dubbed as General) to all the players in an unreliable network. The unreliability of the network is modeled via a fictitious entity called adversary, which can control a fraction of nodes and edges of the network. It is well known that over a completely connected synchronous network, BGP is possible if and only if, the number of nodes controlled by an adversary that can perform arbitrary changes (Byzantine adversary) is less that one third the total number of nodes in the network. Literature has considered various ways in order to circumvent this severe limitation. One of the popular ways is the use of authentication tools (say Public Key Infrastructure and digital signatures) in the protocols for BGP. BGP in the authenticated model is referred as "Authenticated Byzantine Generals" (ABG). It is well known that one can design ABG protocols tolerating any number of Byzantine faults. In the age of modern cryptography, it is reasonable to assume availability of authentication tools such as Public Key Infrastructure(PKI) and digital signatures over any communication network. Owing to the vast improvement in fault tolerance and easy availability of authentication tools, ABG is a popular variant of BGP. Subsequently, it was proved that if two or more instances of any ABG protocol are executed in parallel, any gain in fault tolerance over BGP, is nullified. It is further known that this loss can be prevented by using unique session identifiers in every parallel execution of the ABG protocol.

In this dissertation, we aim to investigate the *advantages and limitations* of using authentication tools in the protocols for BGP. To accomplish the same, we study the problem of ABG under a new fault model whereby the adversary can make some players to act maliciously and read the internal data of some more players. We give the necessary and sufficient condition(s) for designing ABG protocols in the aforementioned fault model. We find that studying ABG in this new fault model leads to *unification* of the extant results on BGP and ABG. Our study can also be seen in the light of fault tolerance of signature schemes for ABG protocols, wherein, even if up to a fraction of the signature schemes become insecure during the execution of an ABG protocol, still the broadcast is guaranteed to succeed.

We also investigate the effect on security of ABG protocols when two or more instances of the same ABG protocol are executed in parallel. We attempt to answer fundamental question such as does there exists a protocol at all? If yes, what is(are) the necessary and sufficient condition(s) for the same ? Our study brings out some interesting results. Contrary to the popular belief, we show that it is *not* always in the best interest of the adversary to use his full powers. Our work also unveils the *ineffectiveness* of unique session identifiers in parallel composition of ABG protocols. This essentially controverts the claim in the extant literature that use of unique session identifiers always achieve their goal of truly *separating* different executions of any protocol.

Contents

List of Figures

Chapter 1

Introduction

Completing a task in a collaborative manner is an important feature of any organized group or society. One often comes across this phenomena in real life in many different forms. To name a few, consider (a) a bunch of ants living together in a ant hill (b) a set of armed men guarding a building, (c) members of a family trying to meet ends, and, (d) bunch of robots in a assembly line production. In each of the above mentioned examples, for a successful accomplishment of the job, it is important that every member of the group has a well defined task which he or she is supposed to do. Dividing a task into sub-tasks and assigning each of the sub-tasks to one(or more) member(s) can be easily achieved in two ways: (a) a predesignated entity does the division and conveys the same to each member of the group. This "entity" could either be with in the group or external to the group, or, (b) all the members of the group reach a consensus among themselves and decide their respective responsibilities.

Consider a set of autonomous machines operating in a distributed environment. A trivial example of this would be a set of servers maintaining a distributed database. A frequent task in any distributed database is to execute a transaction. In order to maintain a consistent database, we require that at the end of any transaction, either all the servers commit to the transaction or all of them abort the same. Analogous to the examples given in the previous paragraph, it is imperative that the servers must collaborate among themselves in order to complete the transaction. How hard is it for for the servers to decide on the fate of the transaction ? Trivial – (a) a predesignated machine makes the decision and conveys the same to every other machine and all the machines obey this decision, or, (b) each of the machine sends its opinion to every other machine. The former is essentially achieving a **broadcast** where as later is arriving at a **consensus**. Note that in consensus since, all the machines receive the same data, all of them are guaranteed to get same output as long as each of them applies same decision rule on the data received.

Now consider a scenario where a fraction of the servers develop a bug(hardware/software) where by the "faulty" machines start to behave maliciously i.e. they may be no more consistent in their actions. In case of broadcast, if the predesignated machine itself is faulty, then this faulty machine may as well send different decision to different machines. In case of consensus, "faulty" machines may send different opinions to other machines. In both the scenarios, one can no longer assume that all machines will receive same data. *Thus, in presence of faults, simulating a broadcast or reaching an agreement appears to be non-trivial.*

The problem of arriving at a consensus in presence of faults was formally introduced by Pease *et al.* [PSL80]. Their motivation for the problem stems from a set of sensors on board an aircraft which report the current altitude of the aircraft. However, few of the sensors develop a technical

snag and start reporting different values to different sensors. The problem is then to design an algorithm where by, even if up to a fraction of total number of sensors become faulty, all the non-faulty sensors should agree among themselves on a single value. The problem is popularly known as "Byzantine Agreement" (BA). Pease *et al.* [PSL80] proved that for the agreement to be possible, the number of faulty sensors should be less than one third of the total number of sensors.

To study the phenomena of distributing data in consistent manner across a distributed environment in presence of faults, Lamport *et al.* [LSP82] introduced the problem of Byzantine Generals. The problem is motivated from a setting where by the Byzantine Empire's army headed by a commanding general, wishes to attack an enemy city. The army consists of small units at geographically different locations, each headed by a lieutenant. The general and the lieutenants must decide upon a common action plan by sending messengers to each other. A fraction of the lieutenants are traitors and aim to confuse the loyal lieutenants. It is required that all loyal lieutenants decide upon the same plan of action and a small number of traitors cannot cause the loyal lieutenants to adopt a bad plan. If the traitors succeed in their goal, any resulting attack is doomed. The problem then is to design an algorithm where by all loyal lieutenants adopt the same plan. This problem is popularly referred as "Byzantine Generals Problem" (BGP). Lamport *et al.* [LSP82] proved that an algorithm for BGP is possible if and only if the number of traitors is less that one third the total number of lieutenants. Note that a solution for BGP essentially ensures that despite faults, an honest general can distribute the data reliably. By reliability we mean that a message sent by the general should correctly reach all the receivers in a guaranteed manner in spite of faults in the underlying network. Therefore, the problem of BGP is sometimes also referred as the problem of **reliable broadcast**.

Reaching consensus and reliably broadcasting data are fundamental tasks in any distributed environment. Thus, both the above mentioned results [PSL80, LSP82] are negative in the sense that, in presence of faults, one needs a reasonably large number of non-faulty machines in order to achieve any of these functionalities. In order to overcome this severe limitation, a lot of work in the past has focused on finding a good and realistic model where by one can achieve better results. One such approach, advocated by Pease *et al.* [PSL80], has been use of authentication tools (such as Public Key Infrastructure(PKI) and digital signatures) in algorithms for BA or BGP. This augmented model is called the *authenticated model* and the problems of BA and BGP in this model are referred as *authenticated Byzantine agreement (ABA)* and *authenticated Byzantine Generals (ABG)* respectively. Subsequently, it well known that algorithms for ABA and ABG can tolerate any number of faults, which is a huge improvement over the fault tolerance bounds of BA and BGP. In the age of modern cryptography, it is reasonable to assume availability of authentication tools such as Public Key Infrastructure(PKI) and digital signatures over any communication network. Owing to the vast improvement in fault tolerance and practicality of assuming availability of authentication tools, authenticated setting is an important model in the area of fault tolerant distributed computing.

1.1 Our Pursuit

In this dissertation we aim to gain a better understanding of the *utility and limitations* of using authentication tools in protocols for BGP. In order to accomplish the same, we study the following (a formal and rigorous definition of each of the following problems is presented in respective chapters respectively):

1. We consider the problem of authenticated Byzantine Generals (ABG) in presence of a mixed

adversary, where by, the adversary can corrupt up to a fraction of player maliciously and can read the internal state" of another fraction of players. We strive to answer the following question "*what is(are) the necessary and sufficient conditions(s) to design ABG protocol(s) tolerating aforementioned mixed adversary?*"

2. We investigate the effect of running multiple executions of any ABG protocol in parallel on fault tolerance of the ABG protocol. We wish to answer the following question "*given a Byzantine adversary, when is it possible to design protocol(s) for ABG that work correctly even if multiple instances of the same protocol are executed in parallel?*"

1.2 Contributions of the Thesis

The contributions of this thesis are many-fold:

1. *Better definition:* As a first contribution of this thesis, we argue that the problem of ABG under the influence of a mixed adversary(characterized by active and passive faults) requires a slight modification in the standard definition of ABG available in the extant literature. Our argument stems from the following observation – Any solution to the problem of ABG aims to simulate a broadcast channel over a point to point network in presence of faults. We show that a protocol that satisfies the extant definition of ABG but does not meet our definition *fails* to simulate a broadcast channel, as originally intended. Therefore, the known definition of ABG is not straight away suitable in our setting. None the less, we essentially use the same principles to define a suitably adapted and faithful definition in our setting.

2. *Complete characterization:* As a second contribution, we give the necessary and sufficient condition(s) for designing protocols for ABG tolerating a mixed adversary. We prove that over a completely connected synchronous network of n nodes, if $n \leq 2t_b + min(t_b, t_p)$, then there does not exist any ABG protocol tolerating a mixed adversary that can corrupt up to any t_b nodes actively and can read the internal states of another t_p players. Further, for $n > 2t_b + min(t_b, t_p)$, we design protocols for ABG and thus prove this bound to be tight. Thus, we comprehensively answer problem 1, section 1.1.

3. *Unification:* In the authenticated model such as ABG, it is assumed that the adversary can forge the signatures of only those nodes under its control. In contrast, the unauthenticated model such as BGP, assumes that the adversary can forge the signatures of all the nodes (including honest nodes). The unauthenticated model can also be visualised as following - nodes are using insecure authentication scheme(s), thus adversary can forge messages on behalf of all the nodes. From this it is evident that it makes sense to study the problem of reliable broadcast given a fraction of honest nodes use insecure authentication scheme(s).

   Motivated from this, we initiate a study on the entire gamut of BGP's in between, viz., the adversary can forge the signatures of up to any t_p nodes apart from controlling up to t_b nodes actively. Thus, BGP ($t_b = t$, $t_p = n - t$) and ABG ($t_b = t$, $t_p = 0$) can be seen as mere two extreme points of this entire gamut. Our work gives a *characterization for the entire gamut of adversaries* between ABG and BGP. Therefore, in a way our results *unify* the extant literature on BGP and ABG.

4. *Fault tolerance of signature schemes:* In the age of modern cryptography, it reasonable to assume availability of Public Key Infrastructure (PKI) and digital signatures over any communication network. All known PKI and digital signature schemes are usually based on

the conjectured hardness of some problems like integer factorization [RSA78], discrete logarithms [Gam85], permutations [Sha94, Sha85], lattice based problems [GPV08, Reg04] to name a few. Further, the proofs of the hardness of these problem seem to be beyond the reach of contemporary mathematics. It is widely believed that that for majority of these problems eventually it will be shown that the hardness is indeed true. However, nothing can be said until it is formally established. Thus, it may well be the case that some of these schemes are actually proven to be insecure in due course of time.

An elegant way to deal with this scenario can be the approach adopted by *robust combiners* [MPW07, HKN$^+$05, MP06]. Informally, a (k, n)-robust combiner is a construction that takes in n candidate protocols for a given functionality and combines them into one scheme such that even if up to any k of the n protocols are rendered incorrect during actual execution, the combined scheme is guaranteed to correctly implement the desired functionality. Note that different sets of up to any k candidate protocols may fail during different executions/inputs.

In context of the authenticated model such as ABG, different players may use different signature schemes. Of these various signature schemes used, some of the schemes may well prove to be insecure. Analogous to the philosophy of (k, n)-robust combiner, one will prefer to design protocols for ABG that work correctly even if up to a fraction of signature schemes are rendered insecure. We capture this failure of signature schemes in protocols for ABG by assuming that adversary can forge signatures of up to another t_p players. Thus, t_p can also be seen as a *robustness parameter of authentication* for protocols solving ABG. Therefore, problem 1, section 1.1 can also be seen as an attempt to study $(t_p, n - t_b)$-robust combiner for ABG protocols.

5. *Proof technique:* An integral part of literature on BA/BGP is to answer the following question "when is it possible to design a protocol for BA/BGP within a given model ?" i.e. giving upper bounds on the fault tolerance. A popular technique to prove the impossibility of any protocol solving BA/BGP for a given condition is *system based impossibility proofs*, developed by Fischer *et al.* [FLM85]. All known proofs in the literature based on this technique use *undirected* systems [FLM85, LLR02, FM00b, CFF$^+$05, LLR06].

   For the impossibility proofs presented in this thesis, we conjecture that undirected systems *do not suffice*. Curiously, *directed systems* seem to work. The existing proof technique requires slight modification before being used for directed systems. Thus, our work extends the proof technique developed by Fischer *et al.* [FLM85]. We remark that for open problems such as BA over directed networks, use of the extended technique appears to be imperative.

6. *Composition of ABG protocols:* From the results of Pease *et al.* [PSL80], it is well known that protocols for ABG can tolerate any number of faults. However, Lindell *et al.* [LLR02, LLR06] proved that for $n > t$, protocols for ABG fail to remain secure even when two instances of the same protocol are executed in parallel. They went on to prove a stronger result that for $n < 3t$ there *cannot* 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 $n > 3t$. The result essentially implies that under parallel composition, power of authentication is rendered useless. 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.

We argue that the results in the state-of-the-art implicitly assumes that *if the adversary corrupts(in Byzantine fashion) a player in any one of the parallel executions of the ABG protocol, then the adversary will also corrupt(in Byzantine fashion) the same very player in every other parallel execution(of the given ABG protocol) too.* Clearly, this is a very strong presumption with respect to adversary's behaviour. We study the problem of composition of ABG protocols in the absence of aforementioned assumption. For this, 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 for any number of parallel executions.

7. *Answer open questions:* Lindell *et al.* [LLR02, LLR06] raise the question of finding a realistic computation model for ABG that does allow parallel and concurrent composition for n/3 or more corrupted parties - 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 unique session identifiers which might only make it more "unrealistic".

8. *Power of unique session identifiers in composition of protocols:* Literature [Can01a] advocates use of unique session identifier to *separate* a protocol's execution from its environment. Our results show that use of unique session identifiers help to increase the fault tolerance of protocols for ABG under parallel composition from $n > 3t$ to $n \geq 2t$. Note that stand-alone ABG is possible for $n > t$.

   Thus surprisingly, we find that unique session identifiers may *not always* achieve their goal of truly *separating* the protocol's execution from its environment. However, for most functionalities, unique session identifiers indeed achieve their goal, as is obvious from Canetti's universal composition theorem [Can01a]. We show that with respect to to ABG this anomaly is due to the fact 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. Thus, our work brings out existence of a problem where unique session identifiers fail to truly *separate* the protocol's execution from its environment. Thus, contrary to the literature, we demonstrate limitations of universal use of unique session identifiers in composition of protocols.

9. *Interesting open problems:* Last but not least, our work raises some interesting open questions. The importance of these questions stems from the fact that, solutions to these question will give us better insights into *use and limitations* of – authentication tool in protocols for ABG and unique session identifiers in composition of protocols.

1.3 Organization of the Thesis

In Chapter 2, we construct a mathematically rigorous model within which we present our work. Chapter 3 presents a brief overview of the literature on the problem of BA/BGP/reliable broadcast. This is followed by an overview of some of the popular techniques in the extant literature used in proving upper bounds of fault tolerance and designing the protocols. In Chapter 4, we study the problem of ABG under a new fault model. Chapter 5 considers the problem of composition of protocols for ABG under parallel executions. The thesis concludes with a discussion and open problems in Chapter 6.

Chapter 2

Abstraction and Modeling

In this chapter we construct a detailed and mathematically rigorous model for the problem of reliable broadcast. For our context, the model should permit us:

- to present the problem statement formally, and unambiguously
- to formally capture the notion of reliability
- to prove impossibility results if any
- to prove the correctness of the proposed protocols
- to compare various solutions

The model presented in this chapter meets all the above stated requirements, thus facilitating a rigorous treatment of all the results in this thesis.

2.1 The Problem: Informal Description

Loosely speaking, the main objective of this work is to solve the following problem:

Problem of Reliable Broadcast

Given a *network* of interconnected computing and routing nodes (called players), one of which is designated as General, if each player is augmented with addition power of *authentication*, is it possible to design a *protocol* to *securely* transmit a message from the General to every other node in the network even if a subset of players are *faulty* ?

In order to formally define the problem, we mathematically model and define each of the italicized term: network, protocol, security, fault and authentication

2.1.1 Modeling the Network

As a prelude to describing a protocol, we need a precise definition of the underlying communication network. The definition should be generic enough to capture (if not all then) most of the communication task involved in a protocol so as to be able to precisely answer the questions we wish to

explore. That is, our definition should abstract out the parameters which influence the answers and leave out those that have no bearing on our results. For our purpose it is important to model inter-player communication, connectivity between players, knowledge about timing of events in the network known a priori to the players. We now elaborate each of the above:

1. *Inter player communication:* Consider a bunch of players who wish to coordinate their actions in order to complete a task but cannot communicate with one and another. In such an scenario players can coordinate only if they receive instructions from an entity outside the player set. Thus in a distributed setting such as ours, any protocol essentially requires players to send and receive information to one and other. This sharing of information can take place in many different ways. Some of the popular ways in the literature via which this can happen are: shared memory, message passing via channels, executing remote procedure calls. For our purpose, we assume all communication takes place via message passing only.

   Another important aspect of inter player communication is the 'kind' of channels available to the players. By kind we intent to answer the following: if a message is sent on the channel how many player(s) receive the same ? In literature typically three kinds of channel are considered: (i) unicast or point to point - each channel is associated with two players. Message sent by one player is received by other. (ii) multicast or hypergraph - each channel is associated with a subset of players. Message sent by one player is received by all the players in the subset. (iii) broadcast - a channel is associated with all the players. Message sent by a player is received by all the players. Formally: let $\mathcal{S}$ represents is the set of senders, $\mathcal{R}$ as set of receivers and $\mathcal{P}$ be set of n players where $\mathcal{S} \subset \mathcal{P}$, $\mathcal{R} \subseteq \mathcal{P}$, then, for unicast we have $|\mathcal{S}|=1$, $|\mathcal{R}|=1$. For multicast, $|\mathcal{S}|=1$, $1 < |\mathcal{R}| < n$ and for broadcast $|\mathcal{S}|=1$, $|\mathcal{R}|=n$. Throughout this thesis we work only with point to point channels. Further, we assume that all the channels are perfect i.e. there is no noise in the channels and player(s) in receiver set receive exactly same as what is sent by the sender.

2. *Inter player connectivity:* Consider two players, a sender and a receiver connected via a directed edge from the receiver to the sender. Now if the sender wishes to send some message to receiver, one needs some other player(s) in the network via which message can be routed to the receiver else it is impossible. From this example it is evident that how players are connected to one and another has an important bearing on the possibility of communication. Typically, literature has considered following three options: (i) Complete connectivity - every player can communicate with every other player. This is modeled by having an undirected edge between every two players. (ii) Partial connectivity - some players cannot communicate directly with some players. However this inability to communicate directly is symmetric i.e. if player a cannot communicate directly with player b, then b also cannot communicate directly with a. This is modeled by having undirected edges between pair of players who can communicate directly. (iii) Directed Connectivity - some player(s) can directly communicate with some other player(s) but vice a versa may not necessary be true. i.e. player a may be able to communicate directly with player b but reverse may not be true. This is modeled by having a directed edge from player a to player b. For the purpose of this work we limit ourselves to only (i).

3. *Timing :* A protocol may use the information available locally with each player regarding the timing of events in the network. We assume that time is divided in small discrete time-periods and that any event of significance takes an integral multiple of the unit time-period to occur. We further assume that the players are aware of the time bounds on the various events i.e. lower bound indicating the minimum time for the event to occur and the upper bound

indicating the maximum time before which the event is guaranteed to have occurred. Based on the upper time bounds on the event of reception of a message, literature has distinguished two distinct types of communication networks: (i) Synchronous - any message sent if successfully delivered, is guaranteed to be delivered with in unit time period i.e. in a successful communication, the recipient is guaranteed to receive it in unit time period. If the recipient does not receive the particular message with in the unit time period, one can safely assume that it will not receive that message any later. This is modeled by assuming that all the players in the network have a common global clock. All messages are sent on a clock 'tick', and are received at the next 'tick'. (ii) Asynchronous - there is no upper bound on time with in which message will be delivered. It is equivalent to say there is no notion of global clock. Furthermore, arbitrary (however finite) time units mat lapse between sending and receipt of a message.

Throughout this work we assume the communication network to be synchronous. With most distributed system, assuming a global clock known to every player locally may seem a far-fetched assumption. We never the less assume so because of the following reasons:

(a) A considerable portion of this work consists of proving some tight impossibility results. Assumption of a global clock only strengthens these results, as the impossibility vacuously holds true in asynchronous settings. This is because, although synchrony is a property of the communication network, it may be considered as a parameter of the adversary i.e. the timing of delivery of messages is in the hand of the adversary. [for a discussion on adversary refer to section 2.1.3]

(b) Our proposed protocols can be easily modified to work in scenarios with timing uncertainties as long as time bounds on events is finite and known a priori, say using wait-and-timeout mechanisms.

To summarize, we model the communication network as an undirected synchronous graph, completely connected, over a set of players i.e. players are modeled as nodes of the graph and point to point channels between players are modeled using undirected edges. Note that in order the complete this definition we need to formally define a player. We now do the same.

2.1.2 Modeling the Protocol

Intuitively, a protocol can be defined as an interaction between a set of players. During the interaction, each player sends some messages, receives some messages and does some local computation. There exist many models in the literature which aim to model a set of interactive players at work. Some of the prominent models are: interactive Turing machine [Gol04b, Gol04a, Can01a, Lin03a], the π-calculus [MPW92], I/O automata [Lyn96]. Among these, Ran Canetti advocates use of interactive Turing machine (ITM) model over other models owing to better suitability to deal with security issues in distributed protocols(the reader may see [Can01a] for further details).

Thus, a set of n players, $\mathbb{P}=\{p_1,p_2,\ldots,p_n\}$ is modeled as a set of n interactive Turing machines (ITMs) ITM_1, $ITM_2 \ldots ITM_n$. Each player is assigned a unique identity i.e. no two players have same identity. This is captured by assuming every ITM_i with an identity tape $tapeid_i$ with its identity written on this tape. We model the execution of the protocol by a player as an execution of a program. Note that a set of players executing a particular protocol does not necessarily imply that all the players run the same program. Thus the identity tape $tapeid_i$ consists of the player's identity in the network. We model the ability of a player to invocate another player with the help of activation tapes. We envision each player p_i to have an one bit write only activation tape $tapewa_i^j$

corresponding to each player p_j connected to it via an outgoing edge in the network i.e. number of write only activation tapes with a player p_i is equal to his outdegree in the network. Further, a player p_i has an one bit read and write activation tape $taperwa_i^j$ corresponding to every other player p_j in the network which can communicate directly to this player i.e. number of read and write activation tapes with a player p_i is equal to his indegree in the network.

Further, we model any input given to player p_i during the course of the protocol via a read only input tape $taperin_i$. Similarly any output generated by player p_i in the protocol execution is modeled by an write only output tape $tapewout_i$. An important aspect of any protocol is the inter player communication. We model this with the help of incoming and outgoing communication tapes. It is evident that the number of incoming and outgoing communication tapes with each player is dependent on the topology of the network. Between every two players who can communicate directly with one another, we consider a *shared* communication tape. Typically, the read only incoming communication tape of the receiver is same as the write only outgoing communication tape of the sender. Thus every undirected edge between two players p_i, p_j in the graph is associated with a pair of shared communication tapes $tapecomm_i^j$ and $tapecomm_j^i$. For player p_i, tape $tapecomm_i^j$ is the write only outgoing communication tape and $tapecomm_j^i$ is the read only incoming communication tape. Symmetrically for player p_j, tape $tapecomm_i^j$ is the read only incoming communication tape and $tapecomm_j^i$ is the write only outgoing communication tape. Further, each player p_i has a random tape $taper_i$.

In literature, modeling a secure protocol as a set of interactive Turing machines as defined above is a popular abstraction technique. To complete the formal modeling of a protocol, we elaborate as to how the communication of a message m from a particular player p_i to another player p_j takes place. The communication in any given synchronized round occurs in two phases: send phase and receive phase.
In the send phase, player p_i does the following two steps (simultaneously):

1. write the contents of message m on the outgoing communication tape $tapecomm_i^j$.
2. activate player p_j by writing a 1 on the activation tape $tapewa_i^j$.

In the receive phase, player p_j once activated, reads the contents of tape $tapecomm_i^j$.

2.1.3 Modeling the Fault

In a classical distributed system, one assumes that the system always works correctly i.e. system is not vulnerable to any bug, external/internal attack and the hardware used in the system is reliable. In designing secure distributed protocols, such an assumption is clearly unjustified. What is a "secure" protocol - intuitively a secure protocol is one which works correctly despite adversity in the runtime environment i.e. a secure protocol should be able to tolerate some amount of *faulty* behaviour.

A natural question at this point is: *what kind of faults and how much faults can a given protocol tolerate ?* In order to answer this question one needs to *qualitatively* define faults and develop a *fault scale* to quantify the *fault tolerance* of protocols. We start by understanding what is a *fault* ? Once we understand what a fault is, a possible fault scale to specify the fault tolerance could be: enumerating the set of all the faults (possibly infinite) that the protocol can tolerate.

As a prelude to answer the question of *what is a fault*, we recollect the definition of reliable broadcast mechanism (section 2.1): *a reliable broadcast mechanism is a mechanism where in the attacker controlling the flow of information is the system via some subset of points cannot prevent the broadcast of a particular message.* Here, "points" refer to either players or channels. Thus,

some of the players/channels may be under the control of the attacker, and thus may be *dishonest* i.e. they may not perform the way they should ideally perform. There could be more than one attacker simultaneously attacking the system. Crudely, a *fault* can be understood as the set of all possible events that potentially hinder the working of the protocol. A fault may be natural such as a natural calamity, a software bug, a hardware failure or deliberate such as an attacker attacking the system, virus, denial of service. We now elaborate on the same.

As mentioned in section 2.1.2, a distributed protocol Π on n players can be visualized as a set of ITMs or programs $\{\Pi_1, \Pi_2 \dots \Pi_n\}$ wherein each player p_i executes the program Π_i respectively. A player is said to be *dishonest* if it violates the security of the protocol. A dishonest player can violate the security in one of following ways:

1. The player may completely *stop* executing anything.
2. The player may do *more than* what it is supposed to do i.e. the player along with running the designated program may run other some other *cheating* program as a background process in order to learn more information than it is supposed to gain.
3. The player may run a *different* program Ψ_i such that $\Psi_i \neq \Pi_i$.
4. Any combination of the above.

We remark that a dishonest player may as well *collude* with other dishonest players to violate the security of the protocol. A *fault* thus can be represented by a set of collusions and the program run by each of the players. Formally,

Observation 1 (Fault) *A fault can be represented as a tuple ($\mathbb{T}$,Ψ), $\mathbb{T} \subset 2^{\mathbb{P}}$ is a partition of $\mathbb{P}$ and $\Psi = \{\Psi_1, \Psi_2 \dots \Psi_n\}$ is a set of programs where the player P_i executes program Ψ.*

The fault tolerance of a protocol can then be expressed as a set of faults that the protocol can tolerate. This formulation though correct is tough to work with as the number of possible programs of a given length are exponential in length and this make specifying the fault tolerance as formulated above impractical if not impossible.

The notion of faults in the literature is modeled by a fictitious entity called *adversary*. All the faults that occur in the system are attributed to this fictitious entity. For a protocol to be deemed secure it is not sufficient to show that the given protocol tolerates all known challenges posed by dishonest players but instead one has to prove that the protocol works no matter what the dishonest players do (of course what dishonest player can do or cannot do is governed by the faults that can occur in the system). This essentially requires to capture the worst scenario that the dishonest players can generate. One intuitively feels that this "worst case" may as well be all dishonest people attacking the system in a coordinated fashion. This is modeled by assuming that all the dishonest players are under the control of a *centralized adversary* and act according to the instructions of this adversary. This is often referred as *colluding adversary*. Note that a colluding adversary itself can be visualized as a distributed algorithm.

In literature many different kinds of adversary has been considered, each modeling a different fault setting. We now elaborate on some of the popular kinds studied in the literature. Some of the important parameters based on which adversary is classified into different categories are:

1. **Number of corrupted players:** The classification is based on which set of players that the adversary can choose to corrupt during execution of the protocol.

(a) *Threshold adversary:* In this model the cardinality of the any set (or subset of it) that the adversary can corrupt at any point of time is limited. An adversary is t-limited if at any given time at most t players are corrupt. For such an adversary the possible set of players that adversary can corrupt is $\binom{n}{t}$. An adversary that can corrupt up to any t players is referred as t-adversary.

(b) *Non-threshold adversary:* In this model, all possible set of players which can be potentially corrupted are enumerated explicitly. This enumeration is referred as *adversary structure*,introduced by [Gen96, HM00]. Note that an adversary structure is a strict generalization of a threshold scheme as any t threshold adversary can always be expressed by a adversary structure where size of each element $\leq t$.

An adversary can always choose to corrupt some but not all the players in a element of adversary structure. This ability of the adversary is captured by assuming *monotone adversary structures*. If $\mathbb{P}$ is a player set, then an adversary structure $\mathbb{A}$ is said to be monotone if the following holds:

$$a \in \mathbb{A} \Longrightarrow \forall\ b \subseteq a : b \in \mathbb{A} \tag{2.1}$$

Only problem with this notion is that specifying monotone adversary structure can become cumbersome owing to size. A work around is to consider *minimal adversary basis* $\bar{\mathbb{A}}$, defined as:

$$a \in \bar{\mathbb{A}} \Longrightarrow \nexists b \in \bar{\mathbb{A}} : b \supseteq a \tag{2.2}$$

Consider a set of players $\mathbb{P} = \{P_1, P_2, P_3, P_4, P_5\}$, then an example of monotone adversary structure could be $\mathbb{A}$=$\{\{P_1\}, \{P_3\}, \{P_5\}, \{P_1, P_2\}, \{P_2, P_3\}, \{P_4, P_5\}, \{P_1, P_3, P_4, P_5\}\}$ and minimal adversary basis for it is $\bar{\mathbb{A}}$=$\{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_3, P_4, P_5\}\}$

2. **Adversarial behaviour:** The behaviour of the players corrupted by the adversary depends on what type the adversary is.

(a) *Semi honest:* Semi-honest adversary only gathers information and does not alter the behaviour of the corrupted players. That is adversary can read the internal state of all the corrupt players including all the message ever sent and received by them and attempts to obtain additional information (that should ideally remain private) not derivable solely from the output of the protocol. It model realistic setting such as one where dishonest players collude and run a cheating program in the background on their combined internal state so as to obtain additional information. Semi-honest adversary is sometime also called "honest-but-curious", "passive" or "eavesdropper".

(b) *Fail-stop:* Players continue to execute the code delegated to them until they "die". This is modeled by assuming that once adversary attacks a player, it stops to execute any code and does not send/receive any message any further. The corruption can occur at any point of time during execution of protocol. Thus a fail-stop player may only send a partial set of messages of all the messages it was supposed to send in the round it fails. Once the player fail-stops, it does not send/receive any message or compute anything through out the rest of the protocol.

(c) *Omission failure:* Here the adversary can not only gather additional information present with a faulty player but as well choose to block the messages from the faulty player at

will. Note that this is different from fail-stop in the sense that adversary may choose not to send messages across different rounds. In fail-stop, once a player crashes it is known that it cannot ever send any other message during the rest of the execution of the protocol.

(d) *Byzantine:* Here the adversary can make the corrupted players to behave in any arbitrary manner. This is modeled by assuming that the adversary can ask the corrupt players to execute a code of his choice which may not even be related to the designated protocol code in any way. This is a very strong model as it encompasses all the previously mentioned adversarial behaviour.

(e) *Covert:* Formalized by Aumann and Lindell [AL07], this tries to model adversarial behaviour where by adversary is usually neither semi-honest or Byzantine but instead willing to cheat only as long as he is sure that he will not be caught cheating. Motivation for this can be found in cases like business, financial, political and diplomatic settings where the companies, institutions and individuals involved cannot afford the embarrassment, loss of reputation etc when being caught cheating.

3. **Corruption strategy:** Based on the strategy used by the adversary in choosing the potential victims, one can classify the adversary in one of the following categories:

   (a) *Static adversary:* A static adversary chooses the set of players to be corrupted just prior to beginning of the protocol i.e the choice of which set to corrupt is independent of the protocol's execution instance.

   (b) *Adaptive adversary:* An adaptive adversary can choose the players to corrupt as the execution of the protocol proceeds. Depending on the role different players are performing in a protocol execution, for the adversary it might be more beneficial to corrupt some players than other players. Adaptive model captures this freedom by allowing the adversary to choose the players depending on the computation time elapsed and information gathered by the adversary so far. Here once a player is corrupted, it remains corrupted for the rest of the computation.

   (c) *Mobile adversary:* This model facilitates the adversary with the option of "hopping" across the players as the execution of the protocol proceeds. This is modeled by assuming that the adversary can "release" a corrupted player during the protocol execution and hereafter corrupt some other player in exchange. Here a corrupted player once released by the adversary is deemed as honest. This model was first introduced by [OY91].

4. **Computational power:**

   (a) *Polynomial time:* Here adversary is assumed to be computationally bounded i.e. adversary can use only those adversarial strategies which can run in (probabilistic) polynomial-time.

   (b) *Computationally unbounded:* In this model the adversary is assumed to be computationally unbounded i.e. anything which is computable can be computed by the adversary.

   This distinction regarding the complexity of the adversary led to two very different models in the area of secure computation: the information-theoretic model [MSA88, CCD88] where results hold unconditionally and do not rely on any hardness assumptions and the computational model [GMW87, Yao82] where results depend on hardness assumption.

5. **Adversarial timing** Introduced by Yao [Yao82], this distinction is based on adversary's ability to read data ahead of honest players.

(a) *Rushing adversary:* A rushing adversary is given the power, during each communication cycle, to first collect the messages addressed to the corrupt players and exploit this information to decide on what the corrupted players should send in the same communication cycle.

(b) *Non-rushing adversary:* A non-rushing adversary cannot base the messages to be sent during a particular cycle on the messages the corrupted players receive during the same cycle.

6. **Control over the communication**

(a) *Secure channel:* Here adversary does not have any control over the channels. That is two uncorrupted players communicate securely without adversary hearing or influencing the communication.

(b) *Insecure channel:* Here the adversary can listen to all the communication that takes place over the channel. However, adversary cannot alter any part of this communication. This is sometimes also called *authenticated* channels.

(c) *Unauthenticated:* Here adversary has full control over the channel. That is apart from listening to communication, adversary can alter the communication as per his wish.

Throughout this thesis we work with threshold, static, computationally unbounded, non-rushing adversary in the presence of authenticated channels.

2.1.4 Modeling the Security

Rigorously demonstrating that a given protocol does its job "securely" is an essential component of cryptographic protocol design. This requires to formulate a precise definition of security that captures the requirements of the task at hand. Once such a definition is in place, one can show that a given protocol meets the definition of security and hence is secure. However formulating appropriate definitions of security has always been a tricky task. The definition should on one hand be rigorous, precise and yet not be over stringent, and on the other hand, it should capture our intuitive understanding of security. Traditionally, this has been handled by specifying a set of conditions, which have to be satisfied by any protocol for a given problem in order to be deemed as secure against any adversarial behaviour under consideration. However, this approach is not satisfactory for the following reasons. First, it may be possible that an important requirement was missed. This is especially true because different applications have different requirements, and we would like a definition that is general enough to capture all applications. Second, the definition should be simple enough so that it is trivial to see that all possible adversarial attacks are prevented by the proposed definition. This approach not only sometimes lead to cumbersome definitions but also tends to altogether miss subtle nuances of security. A paradigm shift from this traditional approach is the "ideal-world" "real-world" simulation technique introduced by Canetti [Can01b]. The central idea is to define an "ideal" world functionality which carries out the task at hand in a secure manner, in the presence of an ideal world adversary. Then show that running a protocol in real world influenced by a real world adversary amounts to *emulating* the ideal world i.e. real world adversary attacking the protocol gains no more than ideal world adversary attacking ideal functionality. The ideal world functionality is usually carried out with the help of an incorruptible "Trusted Third Party" (TTP). (An interested reader is encouraged to refer [Gol04b, Gol04a] for excellent discussions on definitional work on security.)

We remark that throughout this work the word security has been used in the sense of *reliability*. Informally, communication between a (honest)sender and a (honest)receiver over a network is said to be reliable if the message output by the receiver at the end of the communication is same as what the sender desired to communicate, despite the adversary trying to disrupt the same. We do not consider other aspects of security such as privacy, fairness, independence of inputs, to name a few (reader may refer to [Lin02, Can96] for further details). In order to capture our requirements of security, we adopt the well established approach of ideal/real world simulation. At a high level, our definitional frameworks consists of the following - First, we consider an ideal process wherein a trusted third party (TTP) is assumed to exists and is securely connected to all the players and the adversary. The TTP is also modeled as an ITM. Using the TTP, a protocol for the desired functionality is designed whose proof of security is obvious. Next, the protocol execution in the real-life (where no TTP exists) is formally modeled. That is, the interaction among the players and the adversary is precisely defined. Finally, a protocol is said to be secure if for any real-life adversary $\mathcal{A}$ that interacts with the protocol, there exists an ideal process adversary $\mathcal{S}$ (that interacts with the ideal process which uses the TTP) such that no distinguisher can learn whether it is interacting with $\mathcal{A}$ and the players running the protocol or with $\mathcal{S}$ and the players running the ideal process.

In a nutshell, to define security of a protocol for some functionality, it is enough if we precisely define the following:

1. A corresponding ideal process (that captures the security requirements sought).
2. The real protocol execution and its interactions with the adversary.
3. The notion of the two executions being indistinguishable

We defer a formal definition of security for each of the problems considered in this work to subsequent chapters.

2.1.5 Modeling the Authentication

Through out this thesis we work with authenticated model. That is we assume that the players are supplemented with authentication tools. We now formally model an authentication scheme. As a prelude, it is necessary that we understand the need as well as the advantages of using authentication in protocols for distributed settings. This will help us to understand what all aspects of authentication our model should capture.

Consider two players p_i and p_j such that p_i wishes to send a message m to p_j. However, p_i is not connected to p_j directly but via player p_k. If player p_k is controlled by the adversary in Byzantine fashion, then adversary may not send to player p_j what it originally received from p_i but instead may send some other altered or completely unrelated message m^* (adversary may even choose not to send any message). In such a case player p_j has no way of looking at the message m^* and tell whether it is same as what p_i had originally sent or is it different. Consider another similar setting: p_i wishes to send a message m to p_j. p_i is connected to p_j directly as well as via p_k. Player p_k is controlled by the adversary in Byzantine fashion. p_i sends message m along the direct path between p_i and p_j. Simultaneously adversary sends another message m^* ($m \neq m^*$) to p_j claiming m^* to be a legitimate message from p_i. Notice that even tough p_j knows that one of p_i and p_k is lying for sure, p_j cannot deduce which of the two is faulty. Thus it does not know which of the two messages m and m^* should it use.

Use of authentication to circumvent the above stated problems was first advocated by Pease *et al.* [PSL80]. Intuitively, players use authentication to authenticate themselves and their messages.

This restricts the adversary from forging any arbitrary messages on behalf of any honest player. Based on the above discussion, we formulate the desirable properties that any authentication scheme should have so as to be useful in our protocols:

1. By looking at the message, the receiver should unambiguously be able to establish the sender of the message.

2. If the receiver receives a signed message m originating from an honest player, then the original message sent by honest player is indeed m i.e. adversary cannot ever forge signature of an honest player.

3. Adversary can at most forge signature of players it controls.

4. A correctly authenticated message should always by accepted.

Our modeling of authentication scheme is based on work of Lindell *et al.* [LLR02, LLR06]. We assume a trusted preprocessing phase prior to execution of protocol. In such a preprocessing phase, public key infrastructure of signature keys is generated. That is, each player receives his private signing key in addition to verification keys associated with other players. This is modeled by further augmenting every ITM with a setup-tape. Typically, in the preprocessing phase the setup-tape is initialized for each of the ITM with respective private and public keys. We remark that the preprocessing phase is *not* a part of the protocol. This is due to the fact that setting up a set of public keys is nothing but an agreement problem. Formally, we model a signature scheme as a triplet of algorithms (Gen,S,V) where S,V are algorithms for signing and verification of any message. Gen is a probabilistic generator used to generate signature and verification keys (sk,vk) for a particular player (say P_k). Gen is defined as a function: $(1)^n \rightarrow (sk, vk)$. A given signature scheme is said to be a valid scheme if honestly generated signatures are always accepted. Formally, with non-negligible probability, for every message m, $\mathrm{V}(vk,m,\mathrm{S}(sk,m)) = 1$.

Since adversary $\mathcal{A}$ controls some players, it can always generate messages with valid signatures on behalf of these players. However this does not amount to forgery. adversary is said to succeed in forging if it can generate message with valid signature on behalf of an honest player. The fact that adversary can forge signatures of corrupt players is modeled by a signing oracle. In order for $\mathcal{A}$ to succeed, it must generate a valid signature on a message that was not queried to the signing oracle. Formally this is captured by following experiment: The generator Gen is run outputting a pair of keys (sk,vk). $\mathcal{A}$ is given vk and access to signing oracle $S(sk, \cdot)$. At the end of experiment, $\mathcal{A}$ outputs a pair (m^*, σ^*). Q_m captures the set of all the queries $\mathcal{A}$ ever made to oracle $S(sk, \cdot)$. Then, $\mathcal{A}$ is said to succeed, denoted by $succeed_{\mathcal{A}}(sk,vk)$=1, if $V(vk,m^*,\sigma^*) = 1$ holds true and $m^* \notin Q_m$. This essentially captures the following: $\mathcal{A}$ succeeds in generating a message with valid signature without querying its oracle with this message. A signature scheme is said to be existentially secure against chosen-message attack if success probability of the adversary $\mathcal{A}$ in forging a signature is 0. That is, for every adversary $\mathcal{A}$, following should hold:

$$Pr_{(sk,vk)\leftarrow(1)^n}[succeed_{\mathcal{A}}(sk, vk) = 1] = 0 \tag{2.3}$$

The fact that adversary main gain any other information from signing oracle and query is modeled by another auxiliary information oracle $\mathrm{Aux}(sk,\cdot)$. If this additional information amount to fully revealing sk, adversary can easily forge signatures. We wish to model the fact that adversary may receive information connected to sk that is not necessarily limited to valid signatures. However this information does not enable the adversary to forge signatures. Thus, we model $\mathrm{Aux}(sk,\cdot)$ as an

oracle that does not generate valid signatures, but computes some other function of sk and query message. Formally adversary $\mathcal{A}$ has access to two oracles: $S(sk, \cdot)$ and Aux(sk,$\cdot$). Security is defined in same way as done in previous paragraph, only difference being that the adversary is allowed to query Aux with m^*. Formally, we define an identical experiment as in Equation 2.3, except that the $\mathcal{A}$ has oracle access to both S and Aux. An authentication scheme $< (Gen, S, V)$,Aux$>$ is said to existentially secure against generalized chosen-message attacks if for every adversary $\mathcal{A}$, the probability that $\mathcal{A}$ succeeds in outputting a forgery not in Q_m is 0.

We further remark that all known authentication schemes in literature are usually based on the conjectured hardness of some problems like integer factorization [RSA78], discrete logarithms [Gam85], permutations [Sha94, Sha85], lattice based problems [GPV08, Reg04] to name a few, and, are secure only against a computationally bounded adversary. In light of this fact, our assumption about existence of a authentication scheme secure against computationally unbounded adversary may seem far fetched. We remark that the aim of this work is *not* to explore possibility of such an authentication scheme but instead to explore implications of such a (if at all possible)scheme in protocols for BGP. We further state that it is not the first time in literature that such an assumption has been made. Rather, whole of work on ABG including the seminal paper of Pease *et al.* [PSL80] works with an authentication scheme secure against unbounded adversary.

2.2 Assumptions

To complete the modelling, we now list the set of assumptions within which this work has been developed. These in conjunction with the prequel determine the scope of our results. All of these are standard assumptions in the literature. Below we specify only those assumptions are generic and valid throughout the thesis; specific assumptions that are made within some sections/subsections are clearly mentioned as and when necessary.

1. There exist "magical" authentication schemes which are perfectly secure against computationally unbounded adversary. It is magical in the sense that even a computationally unbounded adversary cannot forge signatures on behalf of an honest player(refer to previous paragraph). However, it cannot prevent the adversary from forging signatures of corrupt players under his control.

2. Keys for authentication cannot be generated with in the system itself. It is assumed that the keys are generated during a preprocessing phase, using a trusted system and distributed to players prior to running of the protocol similar to [LLR02].

3. At the end of preprocessing phase, each player get his private signature keys, in addition to $n-1$ verification keys.

4. Each player in the network knows the topology of the network.

5. Every player is fully aware of the player set taking part in the protocol.

6. An honest player executes the code delegated to him with perfect diligence. Specifically, when asked to execute commands like DELETE, whose correctness cannot be verified through external tests.

7. A player can authenticate his messages using his private key only.

8. A faulty player always acts according to the instructions of the adversary.

9. A faulty player hands over the control to an attacker only if the player is corrupted in any of Byzantine, passive or omission failure manner.

10. If adversary gets hold of the private key of some uncorrupted player, adversary can always forge messages on behalf of such a player.

11. There always exist back channels between every two faulty players and between a faulty player and adversary.

We recall from our problem description given in section 2.1, our domain of enquiry is the question of *possibility* and not *feasibility* i.e. does there exist any protocol or not. Thus, we *do not* focus on the efficiency of the protocols.

Chapter 3

Background and Preliminaries

3.1 Reliable Broadcast

Simulating a broadcast channel over a point to point network, in presence of faults is one of the most widely studied problems in theory of distributed computing. The reason for such a vast interest is two fold: (a) as highlighted in chapter 1, it is a fundamental problem in the area of distributed computing (b) many protocols in the area of secure multiparty computation [GMW87, RB89] rely on the presence of a (physical or simulated) broadcast channel. Such protocols require to consistently distribute some data across all the players and hence rely on a broadcast mechanism in the underlying network.

Informally, the problem of reliable broadcast is as follows: Given a network of n players(nodes) $\{P_1, P_2, \ldots P_n\}$, one of which is designated sender $\mathbf{S}$. $\mathbf{S}$ wishes to reliably transmit a message $\mathbf{m}$ to all the other players(nodes) despite faults in the network i.e. at the end of transmission all the players should receive message $\mathbf{m}$. If there exists an physical broadcast channel (such as television or radio networks) that connects all the players, then the problem of reliable broadcast is trivially soluble ($\mathbf{S}$ just needs to transmit the message along that channel for everybody else to receive it). However, setting up a physical broadcast channel among a set of players that wish to interact is for most cases practically not viable. Therefore, such a communication channel has to be *virtually* established via a protocol. In other words, the protocol when executed should be able to simulate the presence of a physical broadcast channel among the given set of players. As highlighted in Chapter 1 and Section 2.1, designing such protocols is the subject of this dissertation.

In order to understand the informal description of the problem statement presented in section 2.1, we need to understand the behaviour of a physical broadcast channel. Consider the following scenario: given a set of n players $\{P_1, P_2, \ldots P_n\}$, connected via a physical broadcast channel $\mathcal{C}$. One of the n players is designated as sender $\mathbf{S}$. $\mathbf{S}$ sends a message $\mathbf{m}$ via $\mathcal{C}$. Further, we assume that a up to any t of the n players are malicious. At the end of broadcast, every player outputs the value it receives from $\mathbf{S}$. By virtue of broadcast channel $\mathcal{C}$, all the n players is guaranteed to receive message $\mathbf{m}$. Thus, all the honest players will output value $\mathbf{m}$. Adversary can make the malicious players to output a message different from $\mathbf{m}$.

We now formally define the problem of reliable broadcast. Since we require any protocol which satisfies our definition to simulate a broadcast channel over a point-to-point network, our definition should match the functionality of a physical broadcast channel. Thus, our definition is motivated from the behaviour of a broadcast channel in presence of faults, as elaborated in the previous paragraph.

Definition 1 (Reliable Broadcast (RB)) *Given a set of n players* $\mathbb{P}$=$\{p_1,p_2,\ldots,p_n\}$*, a finite domain* V*,* $V = \{0,1\}$ *and a predesignated player as sender* **S**. **S** *holds an input value* $x_s \in V$ *and at the end of the protocol every player* $p_i \in \mathbb{P}$ *decides on a value* $y_i \in V$*. A protocol* η *among* $\mathbb{P}$ *achieves reliable broadcast, tolerating* t *corruptions, if for any* t *out of* n*, any* $\mathbb{P}$ *and* V*, at the end of the protocol the following three properties hold:*

- Agreement: *All honest players output the same value, i.e.* $p_i, p_j \in \mathbb{P}$, $y_i = y_j$.
- Validity: *If* **S** *is honest and starts with value* $x_s = v$*, then all honest players decide on the same,* $y_i = v$.
- Termination: *All honest players eventually decide.*

This problem sometimes is also referred as the problem of *secure broadcast*. Note that by virtue of broadcast, the faulty players(hence the adversary) will always come to know the value of the sender **S**, thus by security one implies reliability and not privacy. By reliability we mean that the message m sent by **S** should reach the receivers in a guaranteed manner in spite of adversary's action. We now formally introduce two more very popular problems in the area of distributed computing, namely **Byzantine Agreement** and **Byzantine Generals Problem**. Both these problem are closely related to the problem of reliable broadcast. Informally, the aim of both the problems is to maintain a coherent view of world among all the honest players inspite of faulty players trying to disrupt the same.

3.1.1 Byzantine Generals Problem

To study the problem of reliably broadcast in presence of faults, Lamport *et al.* [LSP82] introduced the *Byzantine Generals Problem(BGP)*. Formally, the problem is defined as:

Definition 2 (Byzantine Generals Problem (BGP)) *Given a set of n players* $\mathbb{P}$=$\{p_1,p_2,\ldots,p_n\}$ *and a finite domain* V*,* $V = \{0,1\}$ *and a predesignated player as General* $\mathcal{G}$*.* $\mathcal{G}$ *holds an input value* $x_s \in V$ *and at the end of the protocol every player* $p_i \in \mathbb{P}$ *decides on a value* $y_i \in V$*. A protocol* η *among* $\mathbb{P}$ *solves BGP, tolerating* t *corruptions, if for any* t *out of* n*, any* $\mathbb{P}$ *and* V*, at the end of the protocol the following three properties hold:*

- Agreement: *All honest players output the same value, i.e.* $p_i, p_j \in \mathbb{P}$, $y_i = y_j$.
- Validity: *If* $\mathcal{G}$ *is honest and starts with value* $x_s = v$*, then all honest players decide on the same,* $y_i = v$.
- Termination: *All honest players eventually decide.*

3.1.2 Byzantine Agreement

One of the basic paradigms employed in the literature to design a reliable broadcast protocol is that of fault-tolerant agreement wherein each of the members of the group begins with a (local) value and at the end of the protocol, all the members of that group (globally) agree on the input of some honest group-member. The problem was first introduced by Pease *et al.* [PSL80] and is formally be defined as:

Definition 3 (Byzantine Agreement (BA)) *Given a set of n players* $\mathbb{P}$=$\{p_1,p_2,\ldots,p_n\}$ *and a finite domain* V, $V = \{0,1\}$. *Every player* $p_i \in \mathbb{P}$ *holds an input value* $x_i \in V$ *and at the end of the protocol decides on a value* $y_i \in V$. *A protocol* η *among* $\mathbb{P}$ *solves the problem of Byzantine agreement, tolerating* t *corruptions, if for any* t *out of* n, *any* $\mathbb{P}$ *and* V *it satisfies the following conditions:*

- Agreement: *All honest players output the same value, i.e.* $p_i, p_j \in \mathbb{P}$, $y_i = y_j$.
- Validity: *If all honest players start with initial value* $x_i = v$, *then all honest players decide on it,* $y_i = v$.
- Termination: *All honest players eventually decide.*

For all the three problems - RB, BGP and BA, from their respective problem definition, it is evident that $|V|$ should be at least 2 else the problems becomes trivial (every honest player simply decides on the only value in V). Similarly, the problems also become trivial if $n-t < 2$ i.e. there is at most one honest player (honest player always decides on his input value). It is also interesting to note that in each of the three problems "validity" and "agreement" conditions are orthogonal. In RB(or BGP), validity can always be achieved by asking the sender **S** (or General $\mathcal{G}$) to send his value to everyone else and every players decides on this value. However, if the sender **S** (or General $\mathcal{G}$) is faulty, then different honest players may agree on different value, thus violating the agreement condition. Similarly, agreement can always be achieved by all honest players deciding on some default value $v_o \in V$, but this does not always satisfies the validity condition. It is easy to see that this orthogonal relation between validity and agreement conditions holds for the problem of BA too. This suggests that the solution to each of the three problem must be somewhere in between the two extremes, involving several (but finite) many rounds of interaction and information exchange between players to satisfy both validity and agreement conditions, resulting in interesting problem. The termination condition refrains any honest player from being perennially in an 'undecided' state.

3.1.3 Relation between RB, BGP and BA

From the definitions of reliable broadcast [Definition 1] and Byzantine Generals problem [Definition 2], it is easy to see that the problem of RB is same as BGP (just interchange the sender **S** and the General $\mathcal{G}$,). That is, a protocol that solves RB will also solve BGP and vice a versa. In literature, the problem of reliable broadcast has been studied as Byzantine Generals problem (BGP). Thus, in line with the literature, for the rest of this work we will also with the definition of BGP. Lamport *et al.* [LSP82] proved that over a completely connected synchronous network, for BGP to be possible among a set of n players tolerating up to t Byzantine faults, $n > 3t$ is necessary and sufficient condition. Surprisingly, Pease *et al.* [PSL80] proved the same fault tolerance bound of $n > 3t$ for the problem of BA.

Subsequently, it is well known that for $n > 2t$, BA is same as BGP. That is, if the majority of players are honest, then a protocol for BA exists if and only if a protocol for BGP exists. The 'equivalence' is based on the following *intuitive* reductions:
(a) *From BGP to BA*: Given a protocol for BA, a protocol for BGP can be constructed by letting the General send his input to all the players who in turn execute a BA protocol over the received values and reach agreement.
(b) *From BA to BGP:* Given a protocol for BGP, a protocol for BA can be constructed by executing n BGP protocols with player i behaving as the General in the i^{th} protocol; finally, each player

outputs the majority of the n outputs obtained from these n BGP protocols.

For $n \leq 2t$, problem of BA itself not defined. This is because of the fact that faulty players being in majority (as compared to honest players) can always ensure that for any protocol for BA, the validity rule is always violated. Note that for $n \leq 2t$ *if* BA is possible then BGP is also possible. However, vice a versa is not true. As a matter of fact it is not known whether possibility of BGP when honest players in minority implies BA or not.

3.2 Related Work

After the seminal papers by Lamport, Shostak and Pease [PSL80, LSP82], the problem of BGP/BA was later studied in many different models leading to a plethora of results. We now give a brief overview of some of the important results in various models.

3.2.1 Synchronous networks

The result of $n > 3t$ given by Lamport, Shostak, and Pease [PSL80, LSP82] holds for synchronous networks, active adversaries, and both unconditional or computational security.

Unconditional security

Active adversary: Pease *et al.* [PSL80, LSP82] proved the impossibility of any perfectly secure protocol solving BGP against active corruption. Later, Karlin and Yao [KY84] generalized this lower bound for (non-perfect)unconditional security. A generic technique to prove impossibility results in this area was given by Fischer *et al.* [FLM85]. The bound $n > 3t$ was shown to be tight by Pease *et al.* [PSL80, LSP82] however, their protocol was inefficient. First efficient protocol for $n > 3t$ was introduced by Dolev and Strong [DS82]. This was followed by a series of efficient protocols [DFF$^+$82, TPS87, BNDDS87, FM97, BGP89, CW92, GM93].

Simultaneously work was done on the round complexity BA/BGP. Fischer and Lynch [MN82] proved that any deterministic protocol for BA will take atleast $t+1$ rounds. The protocol given by Pease *et al.* [PSL80, LSP82] took $t+1$ round, thus proving the bound to be tight. This was followed by a set of of solutions [BGP89] that were sub-optimal in fault tolerance($n > 4t$). Subsequently, Garay and Moses [GM98] gave the first efficient protocol that was both round optimal and optimally resilient($n > 3t$).

Rabin [Rab83a] and Ben-Or [BO90] independently proved that the lower bound of $t+1$ on number of communication rounds does not apply for probabilistic protocols. Ben-Or [BO90] gave a protocol that terminates in a constant expected number of rounds, with resilience $t = O(\sqrt{n})$. Bracha [Bra85] then went on to show the existence of nearly optimally resilient protocols ($n \geq (3+\varepsilon)t$, for any $\varepsilon > 0$) that terminate in an expected number of rounds. The first optimally resilient protocol requiring a constant expected number of rounds was given by Feldman and Micali [FM97].

Some papers [Bra85, FM97] also explored the implications of having a possibility of non-terminating runs. Though such non-terminating runs were not completely ruled out. Further such protocols do not guarantee simultaneous termination all honest players in the same round.

Fail-stop corruption: Dolev and Strong [DS82], Lamport and Fischer [LF82] considered BGP under the influence of fail-stop adversaries. They give efficient, unconditionally secure protocols that tolerate any number of corruptions ($n > t$). However the lower bound for round complexity even for fail-stop faults remains $t + 1$ [DS82, LF82]. Dwork and Moses [DM90] went on to prove

that $t+1$ rounds of communication are necessary so as to guarantee simultaneous termination and error probability 0.

Computational security

The lower bound for rounds remains $t+1$. Here one expects protocols to rely on assumptions of one-way functions and use of digital signatures. Later is popularly referred as *authenticated model* which we will see in detail in section 3.3.

3.2.2 Asynchronous networks

In asynchronous setting, the definition of BGP [definition 2] needs a modification. This is because unlike synchronous model, in asynchronous settings an honest player can never distinguish between the case when the sender is correct but there is an arbitrary delay and where the sender is corrupted and does not transmit a message at all. In order to over this difficulty, Bracha [Bra87] defined a somewhat weaker version of BGP where the adversary can cause non-termination:

Definition 4 (Asynchronous Byzantine Generals Problem (ABGP)) *Given a set of n players $\mathbb{P}$=$\{p_1$,$p_2, \ldots, p_n\}$ and a finite domain V, $V = \{0,1\}$ and a predesignated player as General $\mathcal{G}$. The General $\mathcal{G}$ starts with an input value $x_s \in V$ and at the end of the protocol every player $P_i \in \mathbb{P}$ decides on a value $y_i \in V$. A protocol η among $\mathbb{P}$, solves ABGP, tolerating t corruptions, if for any t out of n, any $\mathbb{P}$ and V, at the end of the protocol the following three properties hold:*

- Agreement: *If any honest player terminates then all honest players also terminate deciding on the same value i.e. $P_i, P_j \in \mathbb{P}$, $y_i = y_j$.*
- Validity: *If the General $\mathcal{G}$ is honest and starts with value $x_s = v$, then all honest players decide on the same, $y_i = v$.*

In contrast to the problem of ABGP, the problem of BA in its original form still makes sense in a asynchronous network since each honest player is guaranteed to *eventually* receive messages from atleast $n-t$ other players. Fischer *et al.* [FLP85] proved that in an asynchronous network, there does not exists any protocol w.r.t to fail-stop corruption that achieves agreement with probability 1 and is guaranteed to always terminate even if a single player is corrupt. This inherently meant all protocols in this will be probabilistic. Some papers [DLS88, MT07] tried to explore the minimum synchrony required to over come the pessimistic result given by Fisher *et al.* [FLP85].

Unconditional security

Active corruption: Ben-Or [BO83] gave the first asynchronous protocol with unconditional security, tolerating $t < n/5$ faults. However the protocol was inefficient but for $t = O(\sqrt{n})$, the protocol is efficient and requires a constant expected number of rounds. For $n > 4t$, Feldman [Fel89] gave the first efficient protocol. Bracha [Bra87] gave the first optimally resilient but inefficient protocol, tolerating $n > 3t$. Canetti and Rabin [CR93] gave the first efficient protocol with optimal resilience, $n > 3t$, requiring a constant expected number of rounds.

Fail-stop corruption: Bracha and Toueg [BT85] formally proved that there cannot exist any protocol tolerating more than $t > n/2$ fail-stop faults. [BO83] gave the first protocol to achieve this bound. However this protocol is inefficient.

Computational security

Refer to section 3.3.

3.2.3 Popular variants

We now give a brief overview of some the popular variants of the BGP/BA.

Firing squad: Consider a player who wishes to unexpectedly start the execution of a new protocol (even in a fully synchronous network). An honest initiator should be able to make all the other honest players start the protocol in the same communication round where as a corrupt player should not be able to initiate a protocol in a way such that not all honest players start simultaneously. Informally stated the goal is to make all the honest players synchronously perform a common action during the same round, although the players initially do not have a common point of time when this action is to be performed. In order to address this, Burns and Lynch [BL87] introduced the *firing squad problem* w.r.t synchronous networks without any common clock with the aim of initializing a global clock among the players so as to achieve full synchronicity. In the same work, Burns and Lynch give an efficient construction to transform any secure protocol for broadcast with $n > 3t$ into a secure protocol for the firing squad problem. Subsequent paper on this is [CDDS89].

Strong validity: Most agreement protocols considered in the literature allow the honest players to agree on a default value in case all honest players do not start with the same input value. Neiger [Nei94] defined the *strong consensus problem* where the finally agreed value must be the input value of at least one honest player. With respect to synchronous networks, an active t-adversary, unconditional security, and an input domain of size $m(|V| = m)$, they proved $n > max(3, m)t$ to be necessary and sufficient condition to solve the problem of strong consensus. [FG03] gave the first efficient protocol for the same.

Weak broadcast: [GP92] introduced the problem of *weak broadcast* where by if the sender is dishonest, it not necessary for all the honest players to output a value. Some or all honest players may not output any value at all. Formally:

Definition 5 (Weak Broadcast (WB)) *Given a set of n players $\mathbb{P}$=$\{p_1$,$p_2, \ldots, p_n\}$ and a finite domain V, $V = \{0, 1\}$ and a predesignated player as sender* **S**. *Sender* **S** *holds an input value $x_s \in V$ and at the end of protocol every player $p_i \in \mathbb{P}$ decides on a value $y_i \in V \cup \{\perp\}$. A protocol η among $\mathbb{P}$, achieves weak broadcast, tolerating t corruptions, if for any t out of n, any $\mathbb{P}$ and V, at the end of the protocol the following three properties hold:*

- Agreement: *If any honest player p_i decides on a value $y_i \in V$, then every other honest player p_j decides on a value $y_j \in \{y_i, \perp\}$.*
- Validity: *If the sender* **S** *is honest and starts with value $x_s = v$, then all honest players decide on the same, $y_i = v$.*
- Termination: *All honest players eventually decide.*

Definition 6 (Weak Agreement (WBA)) *Given a set of n players $\mathbb{P}$=$\{p_1$,$p_2 \ldots p_n\}$ and a finite domain V, $V = \{0, 1\}$. Every player $p_i \in \mathbb{P}$, starts with an input value $x_i \in V$ and and at the end of the protocol decides on a value $y_i \in V \cup \{\perp\}$. A protocol η among $\mathbb{P}$, solves WBA, tolerating t corruptions, if for any t out of n, any $\mathbb{P}$ and V it satisfies the following conditions:*

- Agreement: *If any honest player p_i decides on a value $y_i \in V$, then every other honest player p_j decides on a value $y_j \in \{y_i, \perp\}$.*
- Validity: *If all honest players start with initial value $x_i = v$, then all honest players decide on it, $y_i = v$.*
- Termination: *All honest players eventually decide.*

Mixed adversary: Meyer and Pradhan [MP91], and Garay and Perry [GP92] considered a model where active and fail-stop corruption can occur simultaneously, i.e., that the adversary may corrupt some of the players actively and some other (distinct) players in a fail-stop manner. They formally proved that $n > 3t_b + t_p$ is necessary and sufficient condition for possibility of BGP tolerating a (t_b,t_p)-adversary where adversary can corrupt t_b players actively and another t_p players passively. The importance of their result is the fact that it unifies results of BGP for Byzantine as well as fail-stop adversary.

Multivalued agreement: Turpin and Coan [TC84] introduced the notion of agreement for any finite domain V where $|V| > 2$. A simple way to solve this problem is to encode elements from V in binary and to run $\lceil log_2|V| \rceil$ binary BA [PSL80, LSP82] protocols in parallel, one for each bit. Turpin and Coan [TC84] gave a protocol which requires at most 2 more communication rounds than the binary protocol and an overhead in the over all message complexity of $n^2 log_2|V|$ bits over the binary protocol.

Extended adversary models: Hirt and Maurer [HM00] introduced the notion of a general adversary with respect to secure multi-party computation. They implicitly proved that unconditionally secure broadcast against an active adversary is possible if and only if no three elements of the adversary structure cover the full player set.

There exists a vast literature on the problem of BGP/BA. It is beyond the scope of this thesis to cover entire work in the literature. A very partial list of works includes [Coa87, PP05, BGP92a, BGP92b, CMS89, Lam83, Had83, HH93, HH91, BDP97, BGP89, DRS90, GP90, FM00b, FM00a].

3.3 Authenticated Byzantine Generals

The bound of $n > 3t$ for BGP/BA is a pessimistic result in the sense that no (perfect) protocol can tolerate more that one third of faults for reaching consensus which is central to any task in distributed computing. To over some this severe limitation, Pease *et al.* [PSL80] proposed the use of authentication, where by players are supplemented with authentication tool (such as Public Key Infrastructure and digital signatures) to authenticate themselves and their messages, to thwart the challenge posed by Byzantine players. The augmented problem is popularly known as *authenticated Byzantine Generals* (ABG), *authenticated Byzantine agreement* (ABA) respectively. The problem definition of ABG is same as that of BGP, just that players are supplemented with additional power of a secure authentication scheme with the help of Public Key Infrastructure (PKI) and digital signatures. Formally,

Definition 7 (Authenticated Byzantine Generals Problem (ABG)) *Given a set of n players $\mathbb{P}$=$\{p_1,p_2,\ldots,p_n\}$, each augmented with a secure authentication scheme. Let $\mathcal{G}$ be a predesignated player and V be a finite domain, $V = \{0,1\}$. $\mathcal{G}$ holds an input value $x_s \in V$ and at the end*

of the protocol every player $p_i \in \mathbb{P}$ decides on a value $y_i \in V$. A protocol η among $\mathbb{P}$ solves ABG, tolerating t corruptions, if for any t out of n, any $\mathbb{P}$ and V, at the end of the protocol the following three properties hold:

- Agreement: *All honest players output the same value, i.e. $p_i, p_j \in \mathbb{P}$, $y_i = y_j$.*
- Validity: *If $\mathcal{G}$ is honest and starts with value $x_s = v$, then all honest players decide on the same, $y_i = v$.*
- Termination: *All honest players eventually decide.*

Similarly, ABA can be defined as:

Definition 8 (Authenticated Byzantine Agreement (ABA)) *Given a set of n players $\mathbb{P}=\{p_1, p_2, \ldots, p_n\}$, each augmented with a secure authentication scheme. Let V be a finite domain, $V = \{0, 1\}$. Every player $p_i \in \mathbb{P}$, holds an input value $x_i \in V$ and at the end of the protocol decides on a value $y_i \in V$. A protocol η among $\mathbb{P}$ solves ABA, tolerating t corruptions, if for any t out of n, any $\mathbb{P}$ and V it satisfies the following conditions:*

- Agreement: *All honest players output the same value, i.e. $p_i, p_j \in \mathbb{P}$, $y_i = y_j$.*
- Validity: *If all honest players start with initial value $x_i = v$, then all honest players decide on it, $y_i = v$.*
- Termination: *All honest players eventually decide.*

3.3.1 Related Work

Pease *et al.* [PSL80] formally proved that over a completely connected synchronous network of n nodes, with augmentation of authentication, fault tolerance of protocols for BGP/BA can be amazingly increased to $n > t$ which a vast improvement over the bound of $n > 3t$ for same functionality without authentication. Intuitively, the reason for this improvement can be understood from the fact that in BGP/BA, the faulty players can modify and send messages on behalf of any player. Thus an honest player on receiving two different messages originating from a player can never be sure as to whether some player in between was faulty and altered the message or the original sender of the message itself was faulty and sent different values to different players. From this it is evident that use of authentication is bound to help the protocol. This is because if the sender of honest, and signed the message before sending, then no matter what the faulty players do they cannot introduce a different message in the network on behalf of an honest player. The faulty players can *at most* block the messages from any honest player. Adversary can introduce new values only on behalf of faulty players. Therefore in some sense, use of authentication reduces the problem to somewhere in between Byzantine faults and fail stop faults. Owing to the drastic improvement in the fault tolerance bounds, ABG/ABA are two very famous problems in the area of distributed algorithms. We now present a brief literature survey on the same.

3.3.2 Synchronous network

Unconditional Security

Active corruption: As evident from the discussion in the previous paragraph, it is not a surprise that the fault tolerance bound is same for problem of BGP/BA with fail-stop faults and ABG/ABA

under influence of Byzantine faults. Though unproven, it is easy to see that the equivalence holds for incomplete graphs too. One can easily show that ABG/ABA tolerating a t-adversary over any network $\mathcal{N}$ of n nodes is possible if and only if $n > t$ and $\mathcal{N}$ is $t+1$ connected. Dolev and Strong [DS83] presented first efficient (deterministic) protocol for ABG requiring $t+1$ rounds of communication thereby confirming the usefulness of authentication in both possibility as well as feasibility of distributed protocols. In the same work, Dolev and Strong further proved $t+1$ as the lower bound on the number of rounds for deterministic protocols. Their protocol requires a total number of $O(nt)$ messages. The lower bound for number of round was independently proven by DeMillo *et al.* [DLM82].

Typically in protocols of ABG, players sign their messages before sending. Signing messages is a time consuming process. Motivated from this, [Bor95] investigate the fault tolerance properties of authenticated protocol which require as few signatures as possible. They assume that there are some rounds in the protocol execution where no player signs his messages. They prove that for a bound of $n > 2t$, one needs $log_2(\frac{n}{2}+1)$ rounds. They further prove that in order to tolerate more faults, one requires about two authenticated rounds per additional faulty node. [Bor96b] try to strike a balance between low message complexity and high fault tolerance of authenticated protocols with fast message generation of non-authenticated protocols. They deign efficient hybrid protocols to achieve the same. [Bor96a] argues that key distribution is major hindrance in the protocols for ABG. They argue their case by the fact that all known agreement protocol using message authentication require a complete agreement on all public keys. Because of this, any pre-agreement has to rely on techniques outside the system (e.g. trusted servers which never fail), it is useful to consider lower levels of key distribution which need as few assumptions as possible. They define different levels of authentications such as (1) No authentication (2) Local authentication (3) Crusader authentication (4) Partial authentication (5) Complete authentication and derive bounds for the same. [ST87] try to strike a balance between the simplicity of authenticated protocols without the overhead of signing. They propose techniques to simulate authenticated messages by non-authenticated sub-protocols. This permits to transforms authenticated protocols easily into non-authenticated protocols while retaining some of their properties. However, one loses the fault tolerance properties in the process. Some other works that explore use of authentication in protocols of distributing computing are [KK07, GLR95, SW04].

Lindell *et al.* [LLR02, LLR06] introduced the problem of composition of authenticated Byzantine agreement. Surprisingly they prove that if $n \leq 3t$, there does not exist any protocol for ABG that can compose in parallel even twice. The impossibility arises due to ability of the adversary to borrow messages from one execution and use the same in other execution. They further prove that protocol for ABG compose in parallel for any number of concurrent execution if and only if $n > 3t$. 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 concurrent executions can be designed tolerating $t < n$ faults.

Computational Security

Rabin [Rab83a] presented the first efficient probabilistic protocol. It requires an expected constant number of rounds and tolerates $t < n/4$ player corruptions. Feldman and Micali [FM85] constructed an efficient protocol tolerating $t < n/3$ faults and running in constant expected time. However, their solution requires a one-time interactive pre-computation phase with $\Omega(t)$ rounds of communication.

Bracha [Bra87] proved that, for any $\varepsilon > 0$, there is a protocol that tolerates $t \leq n/(2+\varepsilon)$ faults and runs in an expected number of $O(logn)$ rounds. For the exact bound of $t < n/2$, Toueg [Tou84] gave the construction of a Monte Carlo protocol that terminates in a fixed number of rounds linear

in a customizable security parameter k and guarantees correctness of the outcome with an error probability exponentially small in k. The protocol can be transformed into a protocol of type "Las Vegas" requiring a constant expected number of communication rounds. Where as the protocols in [FM85, Bra87] only assume that a public key infrastructure (PKI) be shared among the players, those in [Rab83b, Tou84] require some additional data to be set up among the players (once for a life time).

3.3.3 Asynchronous network

[BT85, Tou84] formally proved a lower bound of $n > 3t$. Rabin [Rab83b] presented the first protocol for this model tolerating $t < n/10$ corruptions. Toueg [Tou84] followed it up with the first efficient protocol with optimal resilience.

3.4 Proof Techniques

In this section we present a technical overview of few techniques and data structures prevalent in the literature for proving the impossibilities and developing protocols for the problems of BA, BGP and their variants. We present only those techniques that are used in further chapters.

3.4.1 Impossibility Proofs

"There does not exists any protocol among a set of n players, over a fully connected synchronous graph that solves BGP tolerating t malicious players, if $n \leq 3t$". It is evident that the above statement is profound in the sense that no matter what one does, one cannot ever design a (perfect)protocol for BGP under this setting. But how does one prove such a statement ? Just because all known techniques of designing a protocol fails, one may feel the 'impossibility', but this cannot constitute a proof for the same. One of the ways to prove impossibility is to show that no matter what and how much information players share among themselves, one or more properties/conditions of a correct protocol will always be violated. A popular technique in the literature is *bi-valency* argument, introduced by Fischer *et al.* [FLP85]. The technique essentially proves existence of a state from which two different executions will lead to two different decisions. Specifically, for a particular input, one proves that there exists a *bivalent* state prior to the start of the protocol. Then for a particular player who was initially in the bivalent state, one goes on to show that there exist an adversary which can perennially maintain the bivalent state for this particular player. This implies that for the same input, this player at the end of any protocol will give different outputs at different times. This violates the deterministic property of the protocol. The drawback of this technique is that the proofs are cumbersome and not easy to understand.

Consider the following argument for impossibility of BGP [definition 2] over a completely connected synchronous network $\mathcal{N}$ among three players $\mathbb{P} = \{A, B, C\}$ tolerating 1 Byzantine fault (dubbed as 1-out-of-3). We assume their exists a protocol Π that solves BGP for 1-out-of-3 setting. We then show that there exists an input for which adversary can ensure that different honest people will output different values. This violates "agreement" condition of Definition 2. This contradicts our assumption of existence of Π.

In an execution of Π over $\mathcal{N}$, we consider three scenarios α_1, α_2 and α_3 as follows: in α_1 player A is corrupt, C in the General starts with input value 0. In α_2 player B is the General. Adversary corrupts B and makes it interact with A as if B started with input value 0 and interact with C as if B started with input value 1. In α_3 player C is corrupt, B in the General starts with input value 1. We now argue that there exists an adversary which can ensure the following : player C can never

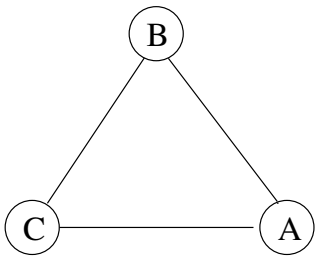

Figure 3.1: Network $\mathcal{N}$ with n=3.

differentiate between scenario α_1 and α_2 (dubbed $\alpha_1 \overset{C}{\sim} \alpha_2$) and player A can never differentiate between scenario α_3 and α_2 (dubbed $\alpha_3 \overset{A}{\sim} \alpha_2$). This is depicted in Figure 3.2.

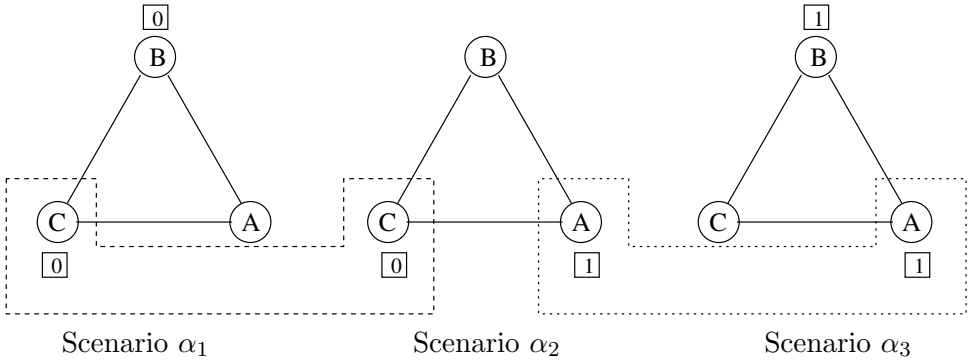

Figure 3.2: C cannot distinguish between α_1 and α_2. Similarly, A cannot distinguish between α_2 and α_3.

As per Definition 2 in scenario α_1, honest players B, C should eventually decide on 0. Similarly, in scenario α_3, honest players A, B should eventually decide on 1. As per our assumption since Π solves ABG, in scenario α_2, honest players A, C should decide on same value. However if adversary can ensure that $\alpha_1 \overset{C}{\sim} \alpha_2$, then player C will decide upon value 0 in α_2. Similarly if adversary can ensure that $\alpha_3 \overset{A}{\sim} \alpha_2$, then player A will decide upon value 1 in α_2. This implies different honest people in α_2 decide on different values, which violates agreement condition of Definition 2. This implies that our assumption of existence of Π is wrong.

To complete the argument all we need to show it that adversary can ensure $\alpha_1 \overset{C}{\sim} \alpha_2$ and $\alpha_3 \overset{A}{\sim} \alpha_2$. We now give adversary for each of the three scenarios. Formally, adversary corrupts player A in α_1 does the following:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. $\mathcal{A}$ sends to player B and C what an honest A would have sent them respectively in α_2.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\alpha_1}(B, A)$ and $msg_i^{\alpha_1}(C, A)$ via A. These are round i messages sent by B and C respectively to A. Players$B$ and C respectively compute these messages according to the protocol run by them and the view they get up to round $i-1$.

In α_2, adversary corrupts player B and does the following:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. $\mathcal{A}$ sends to player A what an honest B would have send to A in α_3 and $\mathcal{A}$ sends to player C what an honest B would have send to C in α_1.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\alpha_1}(A,B)$ and $msg_i^{\alpha_1}(C,B)$ via B. These are round i messages sent by A and C respectively to B. Players$A$ and C respectively compute these messages according to the protocol run by them and the view they get up to round $i-1$).

In α_3, adversary corrupts player C does the following:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. $\mathcal{A}$ sends to player A and B what an honest C would have sent them respectively in α_2.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\alpha_1}(A,C)$ and $msg_i^{\alpha_1}(B,C)$ via C. These are round i messages sent by A and B respectively to C. Players$A$ and B respectively compute these messages according to the protocol run by them and the view they get up to round $i-1$.

Now if we show that no matter for how many number of rounds and no matter what kind of messages are sent in any these scenarios, above mentioned adversary can always ensure that player A in α_2 gets same messages as what A gets in α_1. Then player A has same *view* in α_1 and α_2. Thus A in α_2 decides on same value as A decides in α_1. Similarly, player C in α_2 gets same messages as what C gets in α_3. Thus player C has same *view* in α_3 and α_2. Thus C in α_2 decides on same value as C decides in α_3. This leads to different honest people (A,C) deciding on different values. Using mathematical induction on number of rounds, one can prove that adversary can ensure $\alpha_1 \overset{C}{\sim} \alpha_2$ and $\alpha_3 \overset{A}{\sim} \alpha_2$, no matter for how many number of rounds Π continues to run.

Motivated from this Fischer *et al.* [FLM85] developed an alternative approach. Informally, it starts by assuming there exists a protocol Π for some functionality $\mathcal{F}$ over a particular set of players connected as per a given network $\mathcal{N}$ tolerating t faults. Using multiple(generally two) copies of Π, they construct a system S where by players run Π. It does not make a difference what S solves. All that is known is that S has a well defined behaviour i.e. S has a well defined output distribution. In system S one assumes there are no faults and all players are honest and follow the designated protocol diligently. One then goes on to prove that there exists an input for which system fails to exhibit well defined behaviour. This contradicts the original assumption about existence of a deterministic protocol Π solving $\mathcal{F}$ over $\mathcal{N}$ tolerating t.

We remark that above description is crude and lacks many technical subtleties which are central to the proof technique. To highlight the same, we now formally show (lemma 2) the impossibility of BGP [definition 3] over a completely connected synchronous network $\mathcal{N}$(as shown in figure 3.1) among three players $\mathbb{P} = \{A, B, C\}$ tolerating 1 Byzantine fault (dubbed as 1-out-of-3). This proof was first introduced by [FLM85].

Lemma 2 *There does not exists any deterministic protocol that solves BGP over a completely connected synchronous network $\mathcal{N}$ of $n = 3$ players $\{A,B,C\}$ tolerating 1 Byzantine fault.*

Proof: We assume that there exists a (deterministic)protocol Π that solves BGP among 3 players over $\mathcal{N}$ tolerating 1-adversary. Using two independent copies of Π, we construct a hexagonal system

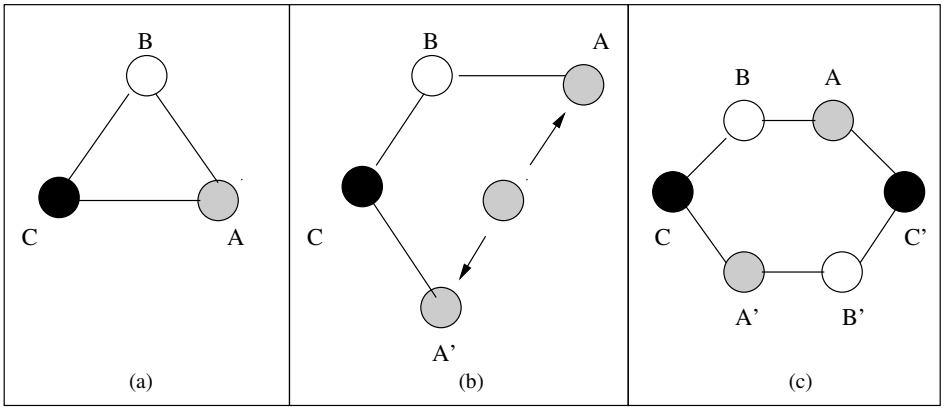

Figure 3.3: Rearrangement of players in proof of Lemma 2

S as shown in Figure 3.3. The proof proceeds by showing that S exhibits contradictory behaviour. This implies that there cannot exist any Π.

We neither know what system S is nor do we know what it solves. All we know is that S is a synchronous system with a well defined output. That is for every input assignment, S has a well defined output distribution. Player A is connected to B and C'; B is connected to A and C; C is connected to B and A'; A' is connected to C and B'; B' is connected to A' and C'. A node a behaving in a Byzantine fashion with a pair of honest nodes, is captured by connecting one of the honest nodes to a and other to a'. a and a' are independent copies of the player a. Each player in S knows only its immediate neighbours and not the complete graph. Also, in reality a player may be connected to either a or a', but it cannot differentiate between the two. It knows its neighbour only by its local name which may be a.

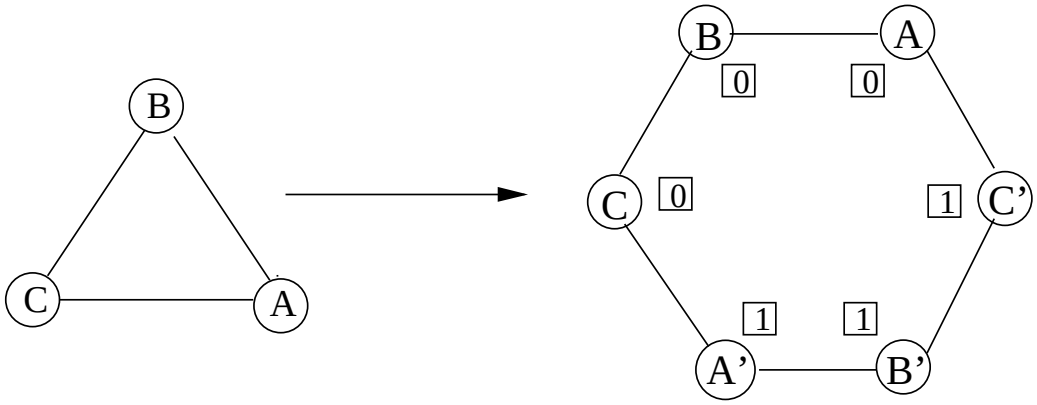

Figure 3.4: Combining two copies of Π to S

Let α_1, α_2 and α_3 be three scenarios in an execution of Π over $\mathcal{N}$, as follows: in α_1 player A is corrupt, C in the General starts with input value 0. In α_2 player B is the General. Adversary corrupts B and makes interact with A as if B started with input value 0 and interact with C as if B started with input value 1. In α_3 player C is corrupt, B in the General starts with input value 1. Further, let α be an execution in S, where each player executes protocol Π starting with the input values as shown in figure 3.4. Each player is α is honest and follows the designated protocol diligently. Here we neither know what system S is supposed to do. Since S does not constitute a 1-out-of-3 BGP setting, therefore definition of BGP [Definition 2] does not tell us anything directly

about the output of players in α. All we know is that S is a synchronous system and Π has a well defined behaviour i.e. players have a well defined output distributions.

Now, consider execution α from the point of view of players B, C. We claim that what ever messages players B, C respectively get in α, adversary can ensure that B, C respectively get same messages in α_1. The claim stems from the following observation: players A, A' form the cutset of S which implies that any message B(or C) gets from any of B' or C' in α *has to necessarily* pass though either A orA'. Note that since A is corrupt in α_1, for any round i, adversary can always send to B in α_1 what A sends to B in round i of α. Similarly adversary can always send to C what A' send to C in α. Since, both B and C start with same input value, execute same code and get same messages in α and α_1, they are bound to get same *view* in α and α_1. This implies player B cannot distinguish between α and α_1 i.e. $\alpha \overset{B}{\sim} \alpha_1$. Similarly $\alpha \overset{C}{\sim} \alpha_1$. This is depicted in figure 3.5 with the help of dotted boxes. Using similar arguments one can prove that $\alpha \overset{A'}{\sim} \alpha_3$, $\alpha \overset{B'}{\sim} \alpha_3$ and $\alpha \overset{C}{\sim} \alpha_2$, $\alpha \overset{A'}{\sim} \alpha_2$.

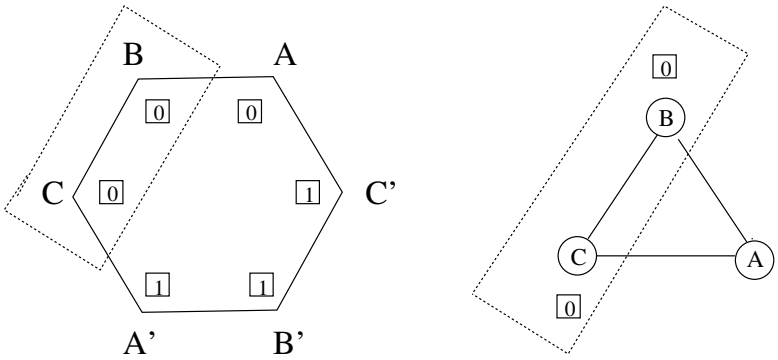

Figure 3.5: Players B, C cannot ever distinguish between α and α_1

Now consider execution α. Players B, C cannot distinguish between α and α_1 i.e. $\alpha \overset{B}{\sim} \alpha_1$ and $\alpha \overset{C}{\sim} \alpha_1$. In α_1, since the General(C)is honest and starts with value 0, as per definition of BGP [definition 2], both B, C will eventually decide on 0. Thus, B,C in α will also eventually decide on value 0 (we are able to make claims regarding player's outputs in α as views of players are same in α and α_1. Thus by analyzing player's outputs in α_1, we can determine their outputs in α.). Similarly, since $\alpha \overset{A'}{\sim} \alpha_3$ and $\alpha \overset{B'}{\sim} \alpha_3$. Thus, A' and B' in α will eventually decide upon value 1. Similarly, since $\alpha \overset{C}{\sim} \alpha_2$ and $\alpha \overset{A'}{\sim} \alpha_3$, C, A' in α will decide on same value as C, A respectively decide in α_2. As per definition of BGP [definition 2], both the honest players will decide on same value in α_2. Then so should C, A' in α. But in α, C has already decided upon 0 and A' has decide upon 1. This then contradicts our original assumption about existence of Π. ∎

3.4.2 Developing Protocols

We now elaborate on some of the popular techniques prevalent in the literature for designing protocols for the problems of BGP/BA.

FloodSet Protocols

The protocol essentially requires each player to 'flood' all the other players with his value. The technique owing to it simplicity is a very popular technique in designing protocols for BGP/BA

and their variants in the presence of fail-stop failures. The technique essentially involves players propagating all the values they have ever seen. To elaborate the same, we give a protocol [Lyn96, page 103] solving BGP [Definition 9] tolerating t fail-stop failures over a completely connected synchronous network of n players. We remark that it is well know that BGP tolerating t fail-stop faults is possible as long as number of honest players is greater than number of faulty ones i.e. $n > t$. We first define the problem statement formally:

Definition 9 (BGP for fail-stop failures) *Given a set of n players $\mathbb{P}$=$\{p_1,p_2 \dots p_n\}$ and a finite domain V, $V = \{0, 1\}$ and a predesignated player as General $\mathcal{G}$. $\mathcal{G}$ holds an input value $x_s \in V$ and at the end of the protocol every player $p_i \in \mathbb{P}$ decides on a value $y_i \in V$. A protocol η among $\mathbb{P}$ solves BGP for stopping failures, tolerating t fail-stop corruptions, if for any t out of n, any $\mathbb{P}$ and V, at the end of the protocol the following three properties hold:*

- Agreement: *All honest players output the same value, i.e. $p_i, p_j \in \mathbb{P}$, $y_i = y_j$.*
- Validity: *If $\mathcal{G}$ is honest and starts with value $x_s = v$, then all honest players decide on the same, $y_i = v$.*
- Termination: *All honest players eventually decide.*

Each player p_i maintains a set W_i which can only contain values from set V. Initially W_i is empty. The General $\mathcal{G}$ sends his value to every player including itself. Every player p_i adds the value that it receives from the General $\mathcal{G}$ to W_i. For, $t+1$ rounds, each player p_i sends W_i to every other player, then adds all the values to W_i that it receives from others in the same round. At the end of $t + 1$ rounds, player p_i applies the following decision rule: If $|W_i| = 1$, p_i decides on the unique element of W_i; else, p_i decides upon the default value v_0. Formally, the protocol is as given in Figure 3.6

FloodSet Protocol

General $\mathcal{G}$ send his value to every player. Player p_i maintains a set W_i. Initially, $W_i = \{\sigma\}$, where σ is the value p_i receives from the $\mathcal{G}$. Then, the protocol is as follows:
Repeat the following steps for rounds 1 to $t + 1$:

1. Send W_i to all the players p_j, $P_j \in \mathbb{P}$.
2. Compute W_i=$W_i \bigcup_j W_j$.

If $|W_i| = 1$ then decide upon v, where $W_i = \{v\}$ else decide upon default value v_0 where $v_0 \in V$

Figure 3.6: A Flood Set Protocol.

In arguing the correctness of FloodSet, we use the notation $W_i(r)$ to represent the set W_i after r rounds. A player is said to be *active* after r rounds if it does not fail by the end of r rounds. As a proof for correctness we prove the following set of lemmas.

Lemma 3 *If no process fails during a particular round r, $1 \leq r \leq t + 1$, then $W_i(r) = W_j(r)$, $\forall P_i, P_j$ that are active after r rounds.*

Proof: Let I be set of players active after r rounds and suppose no player fails in round r. Then since every player in I sends his W to every other player, at the end of round r, $W_i(r) = W_j(r)$, $\forall P_i, P_j \in I$. ∎

Lemma 4 *Suppose that $W_i(r) = W_j(r)$, for all P_i, P_j that are active after r rounds.Then for any round r', $r \leq r' \leq t+1$, then also $W_i(r') = W_j(r')$, for all P_i, P_j that are active after r' rounds.*

Proof: The lemma stems from the fact that players including corrupt players by virtue of always following the designated protocol correctly do not ever introduce a new value in the protocol after round 1. Since $W_i(r) = W_j(r)$, say a player p_k fails in round r', then any value $u \in W_k(r')$, implies $u \in W_i(r')$ and $u \in W_j(r')$. This is because either would have been introduced by p_k or by some other player P_l in round 1. If p_k and/or P_l was alive at the end of round 1, by virtue of FloodSet it implies $u \in W_i(r')$ and $u \in W_j(r')$. If p_k and/or P_l was dead at the end of round 1, since we have $W_i(r) = W_j(r)$ implies $W_i(r') = W_j(r')$, $r \leq r' \leq t+1$. ∎

Lemma 5 *If p_i and p_j are both active after $t+1$ rounds,then $W_i = W_j$ at the end of round $t+1$.*

Proof: Since there can be at most t players who can fail-stop, there must be some round r, $1 \leq r \leq t+1$, in which no player fails. Lemma 3 implies $W_i(r) = W_j(r)$, for all P_i, P_j that are active after r rounds. Lemma 4 implies that $W_i(t+1) = W_j(t+1)$, for all P_i, P_j that are active after $t+1$ rounds. ∎

Lemma 6 *FloodSet solves BGP for stopping failures.*

Proof: Termination is obvious, by decision rule. For validity, let the General $\mathcal{G}$ be honest and start with value v. Then v is the only value that ever gets sent anywhere. Since each player p_i adds initial value to W_i prior to starting of protocol, each $W_i(t+1)$ is non-empty. Since all the players including those that are corrupt do not ever introduce a value different from v, each $W_i(t+1)$ must exactly be equal to $\{v\}$, which also the value decided as per the decision rule. For agreement, let p_i and p_j be two players that decide. This implies P_i, P_j are active at the end of round $t+1$. From Lemma 3.6, $W_i(t+1) = W_j(t+1)$. Decision rule ensures that p_i and p_j decide upon same value. ∎

Protocols using EIG tree

Exponential information gathering is a popular strategy for designing algorithms in the area of BA/BGP. Dubbed as EIG, the strategy requires players to relay all the information they have gathered in each round of communication. Typically this information is their initial values and the values they get from others. The information received by a player along various communication paths is stored in a data structure called *EIG tree*. The technique essentially requires each player to construct EIG tree and apply a common decision rule on the same.

We first elaborate on the data structure EIG tree, introduced by [BNDDS87]. It is a labeled *EIG tree* $T = T_{n,t}$, whose paths from the root represent the chronological order of players along which initial values are propagated. Every chain represented consists of distinct players. If the protocols run for some l rounds then the tree has $l+1$ levels, ranging from level 0 (the root) to level l (the leaves), Typically for most known problems in the area of BA/BGP $l = t+1$, $t+1$ being the round optimality. Each node at level k, $0 \leq k \leq t$, has exactly $n-k$ children. Each node in T is labeled by a string of indices of the players as follows: the root is labeled by an empty string λ. A node with a label $12\ldots k$ has exactly $n-k$ children with labels $12\ldots kj$ where j ranges over {{1 ... n} - {1 ... k}}. As example consider the EIG tree shown if Figure 3.7.

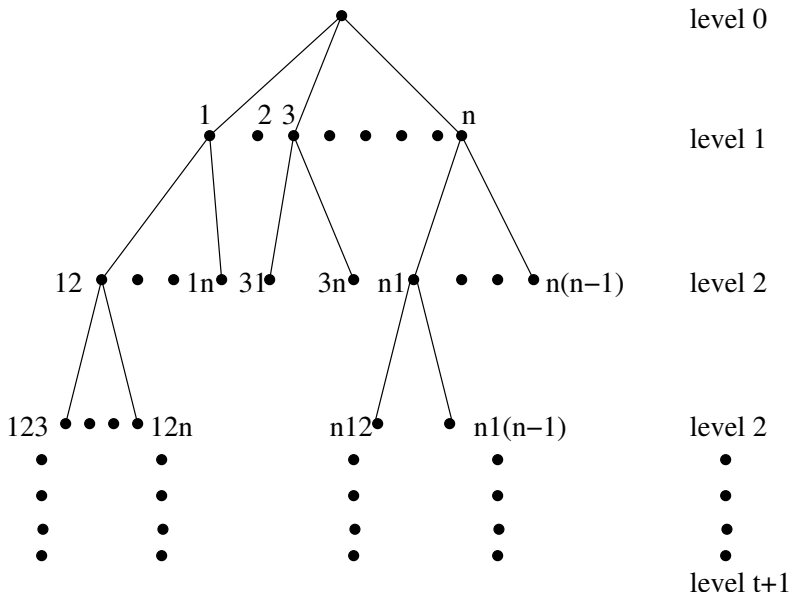

Figure 3.7: *EIG tree* $T_{n,t}$

A player p_i decorates his tree with various values he receives across different rounds of communication during a protocol executions. A node labeled as $12\ldots k$ with a value v in p_i's *EIG tree* means that p_k has told p_i at round k that P_{k_1} had told p_k in round $k-1$ that $\ldots P_1$ has told P_2 in round 1 that P_1's initial value is v. The node may as well be labeled as *null*, which means the communication chain of $P_1, P_2, \ldots, P_k, P_i$ is broken some where due to a failure.

In order to demonstrate the use of *EIG tree*, we present two protocols for BGP. The first protocol, *EIGStop* [Lyn96, Page110] solves BGP [Definition 9] over a completely connected synchronous graph tolerating t fail-stop faults. The second protocol, *EIGByz* [Lyn96, Page119] solves BGP [Definition 9] over a completely connected synchronous graph tolerating t Byzantine faults.Both the protocols assumes that each player p_i in addition to sending messages to other players, can also send messages to itself. This helps in making the algorithm description uniform. These messages in a particular model may not permitted, however these messages can always be simulated locally.

***EIGStop* protocol**
The protocol is given in Figure 3.8. As a prelude to proving the correctness of the protocol we make following observations:

Observation 7 *After $t+1$ rounds of EIGStop algorithm, the following holds:*

1. *$val(\lambda)_i$ is player p_i's input value.*
2. *If xj is a node label and $val(xj)_i = v \in V$, then $val(x)_j = v$.*
3. *If xj is a node label and $val(xj)_i = null$, then either $val(x)_j = null$ or else p_j fails to send a message to p_i in round $|x|+1$.*

Observation 1 and 2 essentially trace the origin of values appearing anywhere in the trees where as third asserts that any value v appearing in the tree must appear in the that tree at some node whose label does not contain index i.

EIG*Stop* Algorithm

General $\mathcal{G}$ send his value to every player. Every player assumes this value from the $\mathcal{G}$ as his input value and runs the algorithm. For every string x that occurs as a label of a node of T, each player has a variable $val(x)$. $val(x)$ is used to store the value with which the player decorates node with label x in his EIG tree. Initially, player p_i decorates the root node of his tree with his input i.e. sets his $val(\lambda)$ to his initial value.

Round 1: Player p_i broadcasts $val(\lambda)$ to all other players including itself. Then, p_i records the incoming information as follows:

1. If value v arrives at p_i from p_j, then p_i sets its $val(P_j)$ to v, else
2. If no value comes or a value outside from V comes at p_i from p_j, then p_i sets its $val(P_j)$ to *null*.

Round k, $2 \leq k \leq t+1$: p_i sends all pairs (x, v) to every other player including itself where x is a level $k-1$ label in T that does not contain index i, $v \in V$, and $v = val(x)$. p_i records the incoming information:

1. If xj is a level k node label in T, where x is a string of player indices and j is a single index, and a message saying that $val(x) = v \in V$ arrives at p_i from p_j, then p_i sets $val(xj) = v$.
2. If xj is a level k node label and no message or a message with a value outside V for $val(x)$ arrives at p_i from p_j, then p_i sets $val(xj)$ to *null*.

At the end of $t+1$ rounds, player p_i applies the following decision rule: let W_i be the set of all the non-*null vals* that ever appear in p_i's tree. If $|W_i| = 1$, then p_i decides on the unique element of W_i; else p_i decides on the default value $v_0 \in V$.

Figure 3.8: *EIGStop* algorithm

Lemma 8 *After $t+1$ rounds of the EIGStop algorithm, the following holds:*

1. *If y is a node label, $val(y)_i = v \in V$, and xj is a prefix of y, then $val(x)_j = v$.*
2. *If $v \in V$ appears in the set of vals of any player, then $\exists i$ such that $val(\lambda)_i = v$.*
3. *If $v \in V$ appears in the set of vals of any player p_i, then there is some label y that does not contain i such that $v = val(y)_i$.*

Proof: 1 follows from from repeated use of observation 7.2. For part 2, suppose $v = val(y)_i$. If $y = \lambda$, we are done. Otherwise, let j be the first index in y. Part 1 then implies that $v = val(\lambda)_j$. For part 3, let v only appear as the *val* for labels containing i and let y be a shortest label such that $v = val(y)_i$. Then y has a prefix of the form xi. but then part 1 implies that $val(x)_i = v$. This contradicts the choice of y. ∎

Lemma 9 *If player p_i and p_j are both honest, then $W_i = W_j$.*

Proof: We assume $i \neq j$ and then show that $W_i \subseteq W_j$ and $W_i \supseteq W_j$.

1. $W_i \subseteq W_j$
   Suppose $v \in W_i$. Lemma 8 implies that $v = val(x)_i$ for some label x that does not contain i. Consider the following two cases:

   (a) $|x| \leq t$.
   Then $|xi| \leq t+1$, so since string x does not contain index i, honest player p_i relays value v to player p_j in round $|xi|$. This implies that $val(xi)_j = v$, so $v \in W_j$.

   (b) $|x| = t + 1$.
   Since there are at most t faulty players and all indices in x are distinct, there must be some honest player l whose index appears in x. Then, x has a prefix of the form yl, where y is a string. From Lemma 8 implies that $val(y)_l = v$. Since l is honest, it would have relayed v to p_j at round $|yl|$. Therefore, $val(yl)_j = v$, so $v \in W_j$.

2. $W_i \supseteq W_j$
   Symmetric to previous case. ∎

Lemma 10 *EIGStop solves BGP for stopping failures [Definition 9].*

Proof: Termination follows from Lemma 9 and decision rule. For validity, If the General $\mathcal{G}$ is honest and starts value v, then from Lemma 9 only values that can ever decorate a player EIG tree is v and *null*. Each W_i is guaranteed to be non-empty as each player starts with value v which implies $val(\lambda) = v$. Thus, each W_i must be exactly equal to $\{v\}$, then as decision rule only output can be v. ∎

***EIGByz* protocol**
The protocol uses same EIG tree data structure $T = T_{n,t}$, as one used in *EIGStop* protocol. Here too propagation strategy is same as used in *EIGStop* protocol. However, decision rule is not same. The protocol is given in Figure 3.9.

***EIGByz* Algorithm**

General $\mathcal{G}$ send his value to every player. Every player assumes this value from the $\mathcal{G}$ as his input value and runs the algorithm. Players propagate values for $t+1$ rounds exactly same as in the *EIGStop* protocol with the exception that if player p_i does not receives a valid message from player p_j, p_i assumes that p_j never sent any message.

At the end of $t+1$ rounds, p_i adjusts its *val* assignment so that any *null* value is replaced by default value v_0. To arrive at a decision, p_i works from the leaves up in its adjusted, decorated tree, decorating each node with an additional *newval*, as follows. For each leaf node labeled x, $newval(x) = val(x)$. For each non-leaf node labeled x, $newval(x)$ is defined to be *newval* held by strict *majority* of the children of node x. If no such majority exists, p_i sets $newval(x) = v_0$. Player p_i decides on $newval(\lambda)$.

Figure 3.9: *EIGByz* algorithm

We prove the correctness of the protocol using Lemmas 11 - 17.

Lemma 11 *After $t+1$ rounds of the* EIGByz *algorithm, the following holds. If i, j and k are all honest players, with $i \neq j$, then $val(x)_i = val(x)_j$ for every label x ending in k.*

Proof: If $k \notin \{i, j\}$, since k is honest, it sends same message to i and j at round $|x|$. If $k \in \{i, j\}$, lemma follows from the convention by which each player relays values to itself. ∎

Lemma 12 *After $t+1$ rounds of the* EIGByz *algorithm, if x is a label ending with index of a honest player, then there is a value $v \in V$ such that $val(x)_i = newval(x)_i = v$ for all honest players p_i.*

Proof: By induction on the tree labels, working from the leaves to top, that is, labels of length $t+1$ to length 1. Suppose x is a leaf, $|x| = t+1$. From Lemma 11, all honest players p_i have same value $val(x)_i$, say v. Then $newval(x)_i = v$.

Inductive step: Suppose $|x| = r$, $1 \leq r \leq t$. From Lemma 11, all honest players p_i have same value $val(x)_i$, say v. Therefore every honest player P_l sends same value v for x to all players in round $r+1$, so $val(xl)_i = v$ for all honest players i and l.

We now show that majority of the labels of children of node x end in indices of honest players. This is because the number of children of x is exactly $n - r \geq n - t$. Since $n > 3t$, this number is always greater than $2t$. Since at most t of the children have labels ending in indices of faulty players, we have the required majority. Then the majority rule ensures that for every honest player p_i, $newval(x)_i = v$. ∎

Lemma 13 *If the General is honest and starts with initial value $v \in V$, then all honest players decide on v.*

Proof: If the General is honest and starts with then every honest player starts with value v. Then all honest players broadcast v in first round, thus $val(j)_i = v$ for all honest players p_i and p_j. Form Lemma 12, $val(j)_i = v$ for all honest players p_i and p_j. Then majority rule implies $newval(\lambda)_i = v$ for all honest players p_i. ∎

To prove agreement [DefinitiondefBGP], we introduce two more terms. First, a subset C of the nodes of a rooted tree is a *path covering* if every path from root to any leaf contains at least one node in C. Second, let α be any execution of the *EIGByz* algorithm. A tree node x is said to be *common* in α if at the end of $t+1$ rounds in α, all honest players have the same $newval(x)_i$. A set of tree nodes is said to be *common* in α if all the nodes in the set are *common* in α. Lemma 12 implies that if p_i is honest, then for every x, xi is a common node.

Lemma 14 *After $t+1$ rounds of any execution α of* EIGByz *algorithm,there exists a path covering that is common in α.*

Proof: Let C be the set of nodes of the form xi, where p_i is honest. As argues above all node in C are common. Now consider any path from the root to a leaf. It contains exactly $t+1$ non-root nodes, and each such node ends with distinct player. Since there are at most t faulty players, there is some node on the path whose label ends in a honest player index. This node must be in C, thus C is path covering. ∎

Lemma 15 *After $t+1$ rounds of* EIGByz*, the following holds. Let x be any node label in EIG tree. If there is a common path covering of the subtree rooted at x, then x is common.*

Proof: By induction on tree labels, working from the leaves up. Let x be a leaf. Then the only path covering of x's subtree consists of single node x itself. So x is common, as required.

Inductive step: Suppose $|x| = r$, $1 \leq r \leq t$. suppose that there is a common path covering C of x's subtree. If x itself is in C , then x in common. Suppose $x \notin C$. Consider any child xl of x. Since $x \notin C$, C induces a common path covering for the subtree rooted at xl. so by the inductive hypothesis, xl is common. since xl was chosen to be an arbitrary child of x, all children of x are common. Then definition of $newval(x)$ implies that x is common. ∎

Lemma 16 *After $t + 1$ rounds of* EIGByz*, the root node λ is common.*

Proof: From Lemma 14 and 15.

Lemma 17 EIGByz *solves BGP as per definition 2 for n players tolerating up to t Byzantine faults, if $n > 3t$.*

Proof: Termination is obvious. Validity follows from Lemma 13. Agreement follows from Lemma 16 and decision rule. ∎

Chapter 4

ABG in Dual Failure Model

4.1 Introduction

A large part of literature in the area of security considers the adversary to have same amount of control over all the corrupt players. This sometimes is referred as *mono-type corruption* or *uniform corruption*. An alternative fault model could be where the adversary does not same amount of control over all the corrupt players. For example – adversary may control some players in active manner, some others in passive fashion, another fraction as fail-stop, so on and so forth. This is popularly known as *mixed adversary*. But why should one study mixed adversary model ? Three major reasons to do so are:

1. *Better modeling:* In many real life settings, processors may not always be prone to same kind of failures/faults. There exists scenarios where processors may exhibits different faulty behaviours such as – some processors may at most fail-stop where as some other may act in malicious fashion. Mixed adversary model helps in adequately capturing this non-uniformity of corruption.

2. *Generalization:* Mixed adversary generalizes the adversary models where by only mono-type(or uniform) corruption is considered. This follows from the simple observation that uniform corruption can always be treated as a special case of non-uniform corruption.

3. *Better insights:* In the area of security the computability/complexity of a task is dependent on the power of adversary. Mixed adversary model helps us to gain a better understanding of the relation between computability/complexity of the task at hand and the adversarial power.

Some of the earliest work that explores mixed adversary model includes [KA94, SKR02, BPC$^+$08, CPA$^+$08, PCSR07]. With respect to BGP, mixed adversary model has been considered in [MP91, GP92, AFM99]. Garay and Perry [GP92] consider a (t_b,t_f)-adversary where by the adversary can corrupt up to any t_b players in Byzantine fashion and another up to any t_f players in fail-stop fashion. Garay and Perry proved that BGP over a completely connected synchronous network $\mathcal{N}$ of n nodes tolerating a (t_b,t_f)-adversary is possible if and only if $n > 3t_b + t_f$. Note that for their result, by using $t_f = 0$ one gets $n > 3t_b$ which the result for BGP under Byzantine faults and using $t_b = 0$ one gets $n > t_f$ which the result for BGP under fail-stop faults. Thus, the results of Garay and Perry *unifies* the work on BGP under Byzantine and fail-stop faults. Later on, Altmann *et al.* extended the work of Garay and Perry for the case of non-threshold adversary characterized by a set Z of pairs (A,F) of subsets of $\mathbb{P}$ where the adversary may select an arbitrary such pair (A_i,F_i) from Z and corrupt the players in A_i actively and fail-corrupt the players in F_i. They proved that

BGP over a completely connected synchronous network $\mathcal{N}$ of n nodes tolerating a Z is possible if and only for no three pairs (A_i,F_i),(A_j,F_j), and (A_k,F_k) in Z, $A_i \cup A_j \cup A_k \cup (F_i \cap F_j \cap F_k)=\mathbb{P}$.

As discussed in chapter 1, 3, BGP and ABG are two important problems in theory of distributed computing. In the authenticated model(ABG), it is assumed that the adversary can forge the signatures of only those nodes under its control. In contrast, the unauthenticated model(BGP) can also be perceived as one where players are using insecure signatures and thus apart from corrupt players, adversary can forge the signatures of all the honest players as well. A valid question at this point would be: *what if apart from corrupt players adversary can forge signatures of another fraction of honest players* ? Motivated from this, we initiate a study on the entire gamut of ABG's in between, viz., a (t_b,t_p)-adversary where adversary can corrupt up to any t_b players actively and forge signatures of up to another t_p players. Note that the solution to the problem of ABG under influence of a (t_b,t_p)-adversary answers the question of simulating a broadcast channel for the entire gamut of adversaries between $t_b = t$ & $t_p = 0$ (ABG) and $t_b = t$ & $t_p = n - t$ (BGP). Thus a solution to the problem of ABG tolerating a (t_b,t_p)-adversary should *unify* the results of BGP($t_b = t$ & $t_p = n - t$) and ABG($t_b = t$ & $t_p = 0$). As a prelude to formal defining the problem statement of ABG under the influence of a (t_b,t_p)-adversary, we introduce the model with in which we work.

4.2 Our Model

We consider a set of n players, fully connected, denoted by $\mathbb{P} = \{p_1, p_2, \ldots p_n\}$. Communication over the network $\mathcal{N}$ is assumed to be synchronous. That is, the protocol 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 players in the same round, (and if necessary perform some more local computation), in that order. During the execution of the protocol, the adversary may take control of up to any t_b players and make them behave in any arbitrary fashion. We model the ability of the adversary to forge signatures of up to another t_p players by assuming that the adversary can corrupt up to another t_p players passively. We refer to such an adversary as (t_b,t_p)-adversary. W.l.o.g we assume that adversary always uses his full power, and hence $t_b \cap t_p = \emptyset$. We further assume that the communication channel between any two players is perfectly reliable and authenticated. We also assume existence of a "magical"[1] (signature/authentication) scheme via which the sender signs the message to be sent. This is modeled by each player having a private key for signing and in addition, public verification keys for all other players[2]. 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_b+t_p) players under its control.

4.3 Problem Statement

Before presenting the problem statement formally, we argue that the existing definition of ABG [Definition 7] is not straight away suitable in our setting of (t_b,t_p)-adversary. None the less, we essentially use the same principles to define a suitably adapted and faithful definition in our setting.

As a prelude, we remark that in literature, a player considered to be *faulty* if and only if that player deviates from the designated protocol. Consequently, a player can be non-faulty in two ways – first the adversary is absent and (therefore) player follows the protocol and second the adversary

[1] Refer to assumption 1, section 2.2

[2] Refer to assumption 2 and 3, in section 2.2

is present passively and (therefore) player follows the protocol. For the rest of the chapter, we refer to the former kind of non-faulty player as *honest* and the latter as *passively corrupt*.

Consider a ABG protocol wherein a player, say P_i, is passively controlled by (t_b,t_p)-adversary. By virtue of passive corruption, the adversary can always forge messages on behalf of P_i (this is because adversary can read (and thereafter use) the private key used by P_i for authenticating its messages). In such a scenario, at the end of ABG protocol, is P_i required to output value same decided upon by honest players ? At a first glance the answer may be NO. The rationale being: P_i has lost his private key to the adversary, therefore, in a way P_i is helping the adversary. Thus, an ABG protocol need not ensure passively corrupt players (such as P_i) to output a value same as honest players. However, in the sequel, we present a series of arguments to demonstrate that *any valid ABG protocol tolerating (t_b,t_p)-adversary is required to ensure that all passively corrupt players output same value as honest players.*

1. *Simulation of broadcast channel:* As highlighted in section 3.1, aim of BGP/ABG is to simulate a broadcast channel over a point to point (unreliable)network. Thus, a correct ABG protocol tolerating (t_b,t_p)-adversary should simulate a broadcast channel. We now investigate the behaviour of a physical broadcast channel under the influence of (t_b,t_p)-adversary – Consider a physical broadcast channel, say $\mathcal{C}$, among a set of n players. Adversary can corrupt upto t_b players actively and another upto t_p players passively. Via $\mathcal{C}$, the General sends his input value $v \in \{0, 1\}$ to all the n players. By property of $\mathcal{C}$, all the n players are guaranteed to receive value v. All honest and passively corrupt players will output v. Adversary can make all the actively corrupt players to output a value of his choice (which may be different from v). It is evident from the above example that for any physical broadcast channel, passively corrupt players will always decide upon a same as honest players. Thus, any protocol aiming to *truly simulate* a broadcast channel in the presence of (t_b,t_p)-adversary, *has to ensure* that all the non-faulty (honest and passively corrupt) players output *same* value.

2. *Authentication is a means, not the end:* The objective of any (valid)BGP protocol is to simulate a broadcast channel from a designated sender to a set of receivers. In order to facilitate this process, authentication is used as a tool in protocols for BGP. Clearly, authentication is a means and broadcast is the end. In such a scenario even if the tools fails to do its job (in the case of passively corrupt players), why should the objective be altered ? In order to fulfill the original objective, all non-faulty(honest and passively corrupt) players must output same value.

3. *Ideal world/Real World:* As discussed in section 2.1.4, a standard paradigm to define any problem in security is the Ideal world/Real World simulation technique. We now show that in the ideal world for ABG in the presence of a (t_b,t_p)-adversary, all non-faulty players always decide on same the value. It then follows that the corresponding ABG protocol in the real world has to ensure that all non-faulty players also decide on the same value.

   Informally, consider a set of n players connected to a Trusted Third Party(TTP). (t_b,t_p)-adversary follows its strategy. W.l.o.g let P_i be a passively corrupt player and P_j be a honest player. The General sends a value to TTP. TTP forwards this value to all the n players. All non-faulty players output the value received from TTP. Thus, in the ideal world, P_i and P_j output same value.

4. *In Continuation of the spirit of BGP:* In the extant literature on BGP, one requires all non-faulty players to agree on the same value [Definition 2]. Recall that literature considers a player as *faulty* if and only if that player deviates from the designated protocol. Consequently

a player can be non-faulty in two ways – first the adversary is absent and (therefore) player follows the protocol and second the adversary is present passively and (therefore) player follows the protocol. In continuation, it is then natural to require all non-faulty players to agree on the same value in any ABG protocol as well. Note that passive control models situations where a player is unaware of the fact that his private key has been compromised. In such a case, it is evident that a protocol that facilitates these players to agree too, if one exists, is preferred.

5. *Motivation from real life:* In order to authenticate important documents, use of physical signatures is a common practice in day-to-day life. Consider a person who forges signature of some other person(s) for an undue benefit/advantage. It is well known that in such scenarios the law penalizes the person committing the forgery and not the victim(s) of the forgery. Analogously, for ABG under the influence of (t_b,t_p)-adversary, passively corrupt players should not be penalized for their signatures being forged by the adversary. Thus, all passively corrupt players should be part of agreement like honest players.

4.3.1 Formal Definition

We now formally define the problem of ABG in the presence of (t_b,t_p)-adversary. We capture the requirements of the task in hand using ideal/real world simulation paradigm (section 2.1.4). We first define the ideal process followed by real process.

Ideal process (Ψ_{ideal})

Participants: Ideal process consists of set $\mathbb{P}$ of n players including the General $\mathcal{G}$, incorruptible TTP (trusted third party) and an ideal process adversary $\mathcal{S}$.

Ideal process (Ψ_{ideal}) execution: We assume that all message transmissions in the following protocol are perfectly secure. The ideal process proceeds as follows:

1. $\mathcal{G}$ sends his input value v to TTP and TTP forwards the same to $\mathcal{S}$.
2. TTP sends v to all the n players and $\mathcal{S}$.
3. All non-faulty 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 $\mathcal{G}$ 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. $IDEAL_{TTP,\mathcal{S}}$ denotes the ensemble $\{IDEAL_{TTP,\mathcal{S}}(v)\}_{v\in\{0,1\}}$.

Real life process ($\Psi_{real}(\Pi)$)

Participants: Real process consists of set $\mathbb{P}$ of n players including the General $\mathcal{G}$ and a real process adversary $\mathcal{A}$.

Real process (Ψ_{real}) execution: Here the players interact among themselves as per a designated protocol Π and the real process adversary $\mathcal{A}$. The real process proceeds as follows:

1. Every non-faulty 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 players under his control) to some(or all) of the players.

3. Non-faulty 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 $\mathcal{G}$ has input v, $\mathcal{A}$ has random coins $r_{\mathcal{A}}$, and $\overrightarrow{r} = r_1, r_2 \ldots r_n$ are the random coins of the 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 10 (ABG_{mix}) *A protocol Π is said to be an ABG_{mix} protocol tolerating a (t_b, t_p)-adversary if for any subsets $I, D \subset \mathbb{P}$ of cardinality up to t_b, t_p respectively (that is , $|I| \leq t_b$ and $|D| \leq t_p$), it holds that for every real process adversary $\mathcal{A}$ that corrupts the players in I and passively controls players in D in $\Psi_{real}(\Pi)$, there exists a ideal process adversary $\mathcal{S}$ in Ψ_{ideal} that corrupts the players in I and passively controls players in D, such that the ensembles $IDEAL_{TTP,\mathcal{S}}$ and $REAL_{\Pi,\mathcal{A}}$ are similar.*

4.4 Some Observations and Definitions

In section 3.4.1 we presented a proof for impossibility of 1-out-of-3 BGP protocol. We now make certain observations in the existing proof technique which are needed for our proofs. As elaborated in section 3.4.1, the technique essentially assumes a protocol(Π) and using two independent copies of Π builds a system. The proof then proceeds to show that the system exhibits a contradictory behaviour. This implies non-existence of Π. We now state our observations:

1. It is not necessary that the system is constructed using the assumed protocol (Π) only. Rather one can as well construct the system using some other protocol η as long as existence of Π implies existence of η.

2. Even if the original network is undirected, the system could be directed.

3. It is not necessary to use only two copies the code. One can as well use more than two copies if required.

4. As a strict generalization of (1), system need not always be constructed using a single protocols. Rather it could as well be a collection of different protocol codes.

For the purpose of this work, observations (1) and (2) suffice. We now formulate a precise definition of the *view* of a player in a particular execution. The proof presented in section 3.4.1 uses the fact that whatever *view* certain players get in a particular execution of the system, adversary can generate same view in a run of the (assumed)protocol with a specific input value. By view we wish to capture all that a player ever gets to see during the entire execution of the protocol. Thus the view of a player is formed by all the messages it ever sends and receives during the execution of the protocol. For the proof presented in section 3.4.1, since the adversary can always send any message on behalf of any player, it never required to formulate a rigorous definition of *view*. However, in case of ABG adversary cannot always send any message on behalf of any player. Thus, in the execution of the system, a player might receive a message such that adversary can never ensure that the same player gets a similar message in an execution of a ABG protocol. Thus it is required to rigorously formulate the notion of *view*.

Let $msg_i^{\Omega}(a,b)_a$ denote the message sent by player a to player b in i^{th} round of execution Ω. The subscript a represents the last player who authenticated the message. W.l.o.g we assume that players always authenticate the message before sending. Then view of a player a during execution Ω at the end of round i, denoted by $view_{a,i}^{\Omega}$, can be represented as collection of all the messages it ever send and receives. Formally:

$$view_{a,i}^{\Omega} = \bigcup_{k}(msg_k^{\Omega}(a,x)_a, msg_k^{\Omega}(x,a)_x),\ \forall k \in \{1 \dots i\},\ \forall x \in \mathbb{P} \tag{4.1}$$

The messages sent by player a in any round i of some execution say Ω depends on the internal state of a just before sending the message. The internal state of a player consists of all the data the player possesses including the code the player is executing. For the case of ABG, the internal state of a player a consists of 4 parameters: input value with which a starts, secret key used by a for authentication, code being executed by a, and messages sent and received by a up to round $i-1$ of Ω. Since the outgoing messages in any round are a function of incoming messages, we can rewrite the equation 4.1 as:

$$view_{a,i}^{\Omega} = \bigcup_{k}(msg_k^{\Omega}(x,a)_x),\ \forall k \in \{1 \dots i\},\ \forall x \in \mathbb{P} \tag{4.2}$$

In order to show that the views of 2 different players a, b running in 2 different executions(of some ABG protocol) Ω, Γ respectively till round i are same, we use the following fact: If both players a, b start with same input, use the same secret key and run same code[3], and if for every round $1 \dots i$ their corresponding incoming messages are same, then their views till round i will also be same. [4] Formally:

$$view_{a,k}^{\Omega} \sim view_{b,k}^{\Gamma},\ \text{iff},\ msg_k^{\Omega}(x,a) \sim msg_k^{\Gamma}(x,b),\ \forall k \in (1 \dots i),\ \forall x \in \mathbb{P} \tag{4.3}$$

4.5 Motivating Example

From the result of $n > t$ [PSL80], one might feel that in the presence of (t_b,t_p)-adversary, $n > t_b$ or $n > t_b + t_p$ (using t_b+t_p=t) is sufficient for possibility of ABG_{mix}. However we show that neither $n > t_b$ nor $n > t_b$+t_p is sufficient to solve ABG_{mix}. We support our claim by studying a simple synchronous network $\mathcal{N}$ consisting of three players (as illustrated in Figure 4.1). We prove that there does not exist any protocol solving ABG_{mix} tolerating a $(1,1)$-adversary over $\mathcal{N}$ consisting of three players $\mathbb{P} = \{A, B, C\}$.

Theorem 18 *There does not exist any protocol that solves ABG_{mix} tolerating a $(1,1)$-adversary over a completely connected network $\mathcal{N}$ of 3 nodes.*

Proof: We assume there exists a protocol Π that solves ABG_{mix} tolerating a $(1,1)$-adversary over a completely connected network $\mathcal{N}$ of 3 nodes. Our proof essentially demonstrates that there exists

[3] As pointed out in observation 2, in order to accommodate the fact that players a, b may even run different codes say θ and θ', we require that the message generated for a given player say C by θ for a given input $\mathcal{I}$ should be same as message generated for C by θ' for same input $\mathcal{I}$.

[4] [FLM85] captured this via **Locality Axiom**. In ABG_{mix} a player may also use its private key to determine the outgoing messages. Thus in case of ABG_{mix}, players having same secret key in both the executions is must.

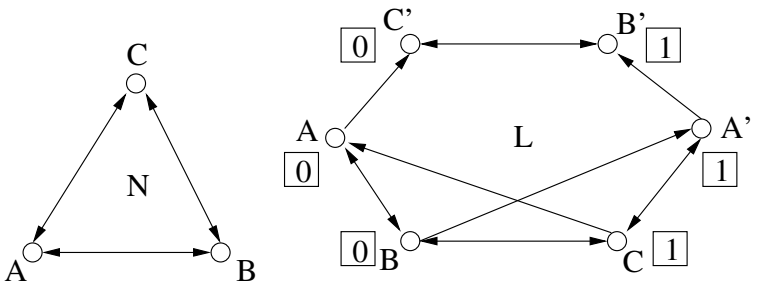

Figure 4.1: Network $\mathcal{N}$ and System L.

an input assignment for which the real process adversary $\mathcal{A}$ (t_b=1, t_p=1) can ensure that non-faulty players do not have a consistent output. In contrast, in the ideal execution all the non-faulty 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 same, thus violating Definition 10.

To show that $\mathcal{A}$ can ensure that non-faulty players do not have a consistent output, we use the proof technique developed by Fischer *et al.* [FLM85] (section 3.4.1). Using Π we create a protocol π'[Definition 11] in such a way that if Π exists then so does π'(Lemma 19). Using two copies of π' we construct a system L (as shown in Figure 4.1). We then show that L must exhibit a contradictory behaviour. This implies impossibility of the assumed protocol Π.

We do not know what system L solves. Formally, L is a synchronous system with a well defined behaviour. That is the system L has a well defined output distribution for any particular input assignment. We show that for a particular input assignment, no such well defined behaviour is possible. Further, no player in L knows the complete system. Each player in aware of only his immediate neighbours. In reality a player may be connected to either a or a', but it cannot differentiate between the two. It knows its neighbour only by its local name which may be a. Specifically, L is constructed in a such a way that whatever messages are sent to some selected players in L, same messages can be ensured by adversary to those very selected players in $\mathcal{N}$. Further, in-neighbourhood of any node a(or a') in L is same as in-neighbourhood of corresponding node a in $\mathcal{N}$.

Let α_1, α_2 and α_3 be three distinct scenarios in execution of Π over $\mathcal{N}$. In α_1, A is the General starting with input 0. Adversary $\mathcal{A}$ corrupts C actively and controls A passively. In α_2, A is the General. $\mathcal{A}$ corrupts A and makes him to interact with B as if A started with input 0, and, interact with C as if A started with input 1. In α_3, A is the General starting with input 1. $\mathcal{A}$ corrupts B actively and controls A passively. Further, let α be an execution of L where each player starts with input value as shown in Figure 4.1. All the players in α are honest and follow the designated protocol correctly.

We claim that no matter for how many rounds Π executes, for any round i, $\mathcal{A}$ can ensure that whatever *view* (as defined in equation 4.2) A, B get in α, $\mathcal{A}$ can generate the same view for A, B in α_1 i.e. $view^{\alpha}_{A,i} \sim view^{\alpha_1}_{A,i}$. This implies that the player A cannot ever differentiate between α_1 and α (dubbed $\alpha_1 \overset{A}{\sim} \alpha$). Similarly, player B cannot ever differentiate between α_1 and α ($\alpha_1 \overset{B}{\sim} \alpha$). From the definition of ABG_{mix} [Definition 10], 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 (We are able to make claims regarding the outputs of A and B in α as their views are same as those in α_1. Thus by analyzing their outputs in α_1, we can determine there outputs in α.). Similarly, $\mathcal{A}$ can ensure that $view^{\alpha}_{A',i} \sim view^{\alpha_3}_{A,i}$ and $view^{\alpha}_{C,i} \sim view^{\alpha_3}_{C,i}$. Thus, $\alpha_3 \overset{A'}{\sim} \alpha$ and $\alpha_3 \overset{C}{\sim} \alpha$. Both A, C in α_3 should decide on value 1. Then so will both A', C in α_3 in α. Similarly, we claim that $\mathcal{A}$ can ensure that

$\alpha_2 \overset{B}{\sim} \alpha$ and $\alpha_2 \overset{C}{\sim} \alpha$. As per the definition of ABG_{mix}, 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 behaviour.

To complete the proof we need to show that $\mathcal{A}$ can always ensure that – A, B get same view in α and α_1, B, C get same view in α and α_2 and A, C get same view in α and α_3. We proof the same in Lemma 21, 23, 25 respectively. As a prelude, we define the protocol π'[Definition 11] and show that if Π exists then so does π'(Lemma 19). ∎

Definition 11 (π') *For players $a, b \in \mathbb{P}$, any statement in Π 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′) [5] in π'. Similarly any statement of the kind "c sends message m to a" in Π is replaced by "c multicasts message m to all instances of a" in π'. Rest all statements in π' are same as those in Π.*

Lemma 19 *If Π exists then π' exists.*

Proof: Implied from Definition 11. ∎

To complete the proof of Theorem 18 we first show that $\mathcal{A}$ can always ensure that A, B get same view in α and α_1. We essentially show that for any round i, $\mathcal{A}$ can always ensure that A, B get same messages in α_1 as A, B get in α. From equation 4.3, it follows that A, B get same view in α and α_1. Informally, that validity of our claim can be seen from the following argument – consider an execution 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 let C follow the designated protocol with input value 1). 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 in Γ A' starts with input value 0 where as in α A' starts with input value 1). 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 L, in α, A' can send a message to A or B only via C. This ensures that in α, any message from A' can reach A or B only after it has been processed by C. Now in α_1, C is faulty and $\mathcal{A}$ controls A passively. Thus whatever C sends to A and B in α, $\mathcal{A}$ can send the same to A and B in α_1.

Thus all we need to show is that whatever messages A receives from B, C in α, $\mathcal{A}$ can always ensure that A gets the same from B, C in α_1 too. Similarly, whatever messages B receives from A, C in α, $\mathcal{A}$ can always ensure that B gets the same in α_1 too. Our technique is as follows – note that what node A receives in round i of α(or α_1) depends on what nodes B and C send to it in round i of α(or α_1). So we need to argue that these messages sent in round i of α and α_1 respectively are same or *can be* made same by adversary. Now the messages B,C send in round i of α and α_1 depend on what they them self receive in previous round $i-1$. This in turn depends on what A, C(or A, B) send to B(or C) in round $i-2$ of α and α_1 respectively. Thus we need to argue that adversary can ensure that whatever messages A, C(or A, B) send to B(or C) in round $i-2$ of α is same as whatever messages A, C(or A, B) send to B(or C) in round $i-2$ of α_1. Note that this continues in a recursive manner until recursion stops at round 1. The entire recursion can be visualized as trees T^A_α and $T^A_{\alpha_1}$ rooted at A for executions α and α_1 respectively as shown in Figure 4.2.

[5] a and a' are independent copies of a with same authentication key.

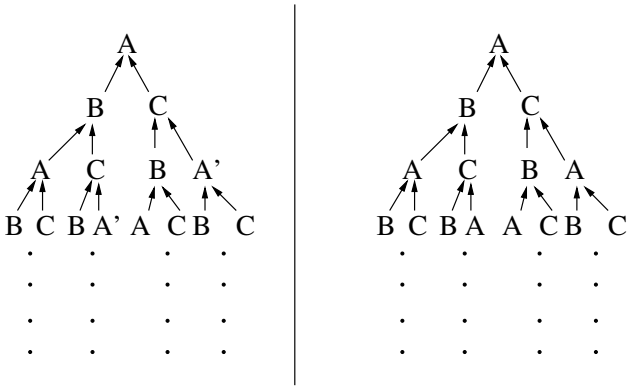

Figure 4.2: T_{α}^{A} and $T_{\alpha_1}^{A}$

We refer to these trees as *execution trees*. We now formally describe execution tree T_{α}^{x}. We name the levels of tree in a bottom up manner. Let the lowest level of tree be 1, next level be 2 and so on. An edge from a node y at level j to another node z at level $j+1$ in the tree represents the message that y sends to z in round j of α. All edges are directed from child to parent and are between adjacent levels only.

For our proof to go through, we require the in-degree for any node y'(or y) in T_{α}^{x} to be same as in-degree of corresponding node y in $T_{\alpha_1}^{x}$. Also, if a node z at level $j+1$ has an incoming edge from node y at level j in $T_{\alpha_1}^{x}$, then correspondingly in T_{α}^{x} node z(or z') at level $j+1$ will also have an incoming edge from node y(or y') at level j. The above two points ensure that structurally both the trees $T_{\alpha}^{x'}$(or T_{α}^{x}) and $T_{\alpha_1}^{x}$ will be exactly same (a node y' in T_{α}^{x} is replaced by its corresponding node y in $T_{\alpha_1}^{x}$). Now consider some node, say b'(or b) at level j in T_{α}^{x}. Then its corresponding node at level j in $T_{\alpha_1}^{x}$ is b. Note that if the messages received by b'(or b) in T_{α}^{x} is same as those received by b in $T_{\alpha_1}^{x}$ and both b'(or b) and b start with same input value, same private key and run same code then both will send same messages to their respective parents in their respective execution trees. One can then use induction of heights of executions trees, say T_{α}^{A} and $T_{\alpha_1}^{A}$, to argue that for any round i, A receives same messages in α and α_1.

For scenario α_1, we now specify the behaviour of the adversary:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. For round 1, $\mathcal{A}$ sends to B what an honest C would have sent to B in execution α_2. For $i \geq 2$, $\mathcal{A}$ authenticates $msg_{i-1}^{\alpha_1}(B,C)_B$ using C's key and sends it to A. For $msg_{i-1}^{\alpha_1}(A,C)_A$, $\mathcal{A}$ examines the message. If the message has not been authenticated by B even once, it implies that the message has not yet been seen by B. Then $\mathcal{A}$ authenticates and sends same message to B as C would have sent to B in round i of execution α_2. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\alpha_1}(A,C)_A$,($\mathcal{A}$ can construct $msg_{i-1}^{\alpha_1}(A,C)_A$, since it passively controls A and has messages received by A in previous rounds.) such that $msg_{i-1}^{\alpha_1}(A,C)_A \sim msg_{i-1}^{\alpha_2}(A,C)_A$, authenticates it using C's key and sends it to B. If the message has been authenticated by B even once, $\mathcal{A}$ simply authenticates $msg_{i-1}^{\alpha_1}(A,C)_A$ using C's key and sends it to B.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_{i}^{\alpha_1}(A,C)_A$ and $msg_{i}^{\alpha_1}(B,C)_B$ via C. (These are round i messages sent by A and B respectively to C). Similarly via A, $\mathcal{A}$ obtains messages $msg_{i}^{\alpha_1}(B,A)_B$ and $msg_{i}^{\alpha_1}(C,A)_C$. (These are also round i messages sent by

B and C respectively to A. Players respectively compute these messages according to their input, secret key, protocol run by them and the view they get up to round $i-1$).

Consider execution α from the perspective of A and B. We now show that messages received by A and B in round i of α are same as messages received by A and B respectively in round i of α_1.

Lemma 20 *$msg_i^{\alpha}(x,A)_x \sim msg_i^{\alpha_1}(x,A)_x$ and $msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_1}(x,B)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.*

Proof: We argue for $msg_i^{\alpha}(x,A)_x \sim msg_i^{\alpha_1}(x,A)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$. Argument for $msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_1}(x,B)_x$ follows similarly. To prove that for any round i, A gets same messages in α and α_1, we use induction on height of T_{α}^A and $T_{\alpha_1}^A$(as shown in Figure 4.2). Only nodes present in T_{α}^A are A, B, C, A'. Corresponding nodes present in $T_{\alpha_1}^A$ are A, B, C, A respectively. Notice that since B' does not appear in T_{α}^A, any A' in T_{α}^A has an outgoing directed edge only and only to C. Similarly, since C' does not appear in T_{α}^A, any A in T_{α}^A has an outgoing directed edge only and only to B.

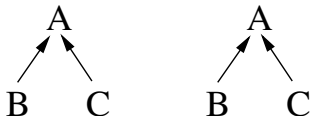

Figure 4.3: T_{α}^A and $T_{\alpha_1}^A$ at the end of round 1.

We analyze the executions trees T_{α}^A and $T_{\alpha_1}^A$ in a bottom up manner. Consider round 1 of executions α and α_1. Consider trees T_{α}^A and $T_{\alpha_1}^A$ at the end of round 1 as shown in Figure 4.3. We claim that A in α and α_1 receive similar messages at the end of round 1. B starts with same input, secret key and executes same code in α and α_1. Thus it will send same messages to A in round 1 of α and α_1 i.e. $msg_1^{\alpha}(B,A)_B \sim msg_1^{\alpha_1}(B,A)_B$. Using aforementioned adversary strategy for α_1, $\mathcal{A}$ can ensure that $msg_1^{\alpha}(C,A)_C \sim msg_1^{\alpha_1}(C,A)_C$. Thus A gets same messages at the end of round 1 in α and α_1.

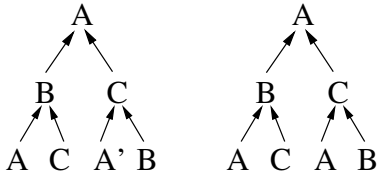

Figure 4.4: T_{α}^A and $T_{\alpha_1}^A$ at the end of round 2.

We now claim that the similarity holds in round 2 as well i.e. $msg_2^{\alpha}(x,A)_x \sim msg_2^{\alpha_1}(x,A)_x$. Consider trees T_{α}^A and $T_{\alpha_1}^A$ at the end of round 2 as shown in Figure 4.4. Node A as well as B start with same input value, secret key and execute same code in both α and α_1 respectively, thus $msg_1^{\alpha}(A,B)_A \sim msg_1^{\alpha_1}(A,B)_A$ and $msg_1^{\alpha}(B,C)_B \sim msg_1^{\alpha_1}(B,C)_B$. Using aforementioned adversary strategy for α_1, $\mathcal{A}$ can ensure that $msg_1^{\alpha}(C,B)_C \sim msg_1^{\alpha_1}(C,B)_C$. Now A and A' start with different inputs thus send different messages to C in round 1. However since A is passively corrupt and C is Byzantine in α_1, $\mathcal{A}$ can construct message $msg_1^{\alpha_1}(A,C)_A$ such that $msg_1^{\alpha_1}(A,C)_A \sim msg_1^{\alpha}(A',C)_A$. Thus C can simulate to receive messages in α_1 same as those in α at the end of round 1. Now B receives same messages in α and α_1 and has same input value, secret key and executes same code, thus $msg_2^{\alpha}(B,A)_B \sim msg_2^{\alpha_1}(B,A)_B$. Using aforementioned adversary strategy $\mathcal{A}$ can ensure that $msg_2^{\alpha}(C,A)_C \sim msg_2^{\alpha_1}(C,A)_C$. Thus $msg_2^{\alpha}(x,A)_x \sim msg_2^{\alpha_1}(x,A)_x$, $\forall x \in \mathbb{P}$ holds.

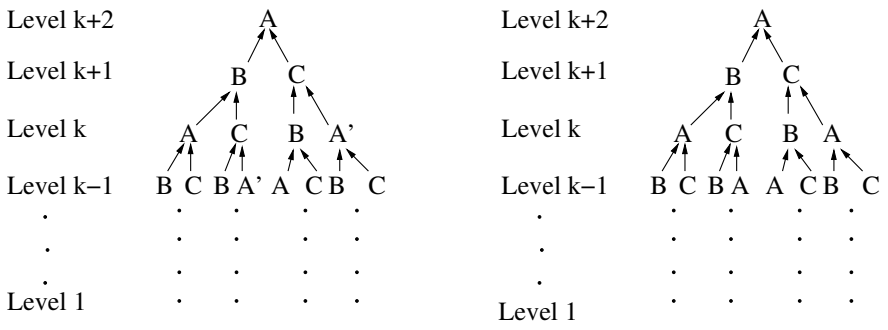

Figure 4.5: T_{α}^{A} and $T_{\alpha_1}^{A}$ at the end of $k+1$ rounds.

Let the similarity be true till some round k i.e. $msg_i^{\alpha}(x,A)_x \sim msg_i^{\alpha_1}(x,A)_x$, $\forall i | 1 \leq i \leq k$, $\forall x \in \mathbb{P}$. We now show that $\mathcal{A}$ can ensure that the similarity holds for round $k+1$ also. Consider T_{α}^{A} and $T_{\alpha_1}^{A}$ at the end of $k+1$ rounds as shown in Figure 4.5. For proving the induction step, we need to show that A at level $k+2$ receives same messages in both trees. Consider edges between level k and $k+1$. From induction hypothesis any node A up to level $k+1$ receives same messages in T_{α}^{A} and $T_{\alpha_1}^{A}$. Since A starts with same input value, secret key and executes same code in both α and α_1 respectively, thus will send same messages in round k i.e. $msg_k^{\alpha}(A,B)_A \sim msg_k^{\alpha_1}(A,B)_A$. Similarly one can argue that $msg_k^{\alpha}(B,C)_B \sim msg_k^{\alpha_1}(B,C)_B$. This is because from the induction hypothesis step on heights of T_{α}^{B} and $T_{\alpha_1}^{B}$, one gets $msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_1}(x,B)_x$, $\forall i | 1 \leq i \leq k$, $\forall x \in \mathbb{P}$. Now consider A' at level k in in T_{α}^{A} and corresponding A at level k in in $T_{\alpha_1}^{A}$. For time being assume A' up to level k in T_{α}^{A} receives same messages as corresponding A in $T_{\alpha_1}^{A}$. Since A' start with different input from A, they send different messages to C in round k. We now claim that $\mathcal{A}$ can ensure that C at level $k+1$ in $T_{\alpha_1}^{A}$ can simulate to receive same message from A' as C at level $k+1$ in T_{α}^{A}. This is because $\mathcal{A}$ controls A passively in α_1, thus can construct messages on behalf of A in α_1. Formally $\mathcal{A}$ can construct $msg_k^{\alpha_1}(A',C)_{A'}$ such that $msg_k^{\alpha_1}(A',C)_{A'} \sim msg_k^{\alpha}(A,C)_A$. Thus C a level $k+1$ receives same messages in both trees. Similarly one can argue that C at level k receives same messages in T_{α}^{A} and $T_{\alpha_1}^{A}$. Since C starts with same input value, secret key and executes same code in both α and α_1 respectively, thus it will send same messages in round $k+1$ to A i.e. $msg_{k+1}^{\alpha_1}(C,A)_C \sim msg_{k+1}^{\alpha}(C,A)_C$. Similarly one can argue that $msg_{k+1}^{\alpha_1}(B,A)_B \sim msg_{k+1}^{\alpha}(B,A)_B$. Thus induction holds for round $k+1$ too. The proof is based on a assumption that A' at level k in T_{α}^{A} receives same messages as corresponding A in $T_{\alpha_1}^{A}$. Note that A' in T_{α}^{A} and A in $T_{\alpha_1}^{A}$ receives messages from B and C. Using induction and arguments similar to those given above one can show that such an assumption indeed holds true. Thus $msg_i^{\alpha}(x,A)_x \sim msg_i^{\alpha_1}(x,A)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$ holds true. ∎

Lemma 21 *$view_A^{\alpha} \sim view_A^{\alpha_1}$ and $view_B^{\alpha} \sim view_B^{\alpha_1}$*

Proof: Follows from equation 4.3 and Lemma 20. ∎

Adversary for α_2:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. For round 1, $\mathcal{A}$ sends to B what an honest A would have sent to B in execution α_1. Similarly $\mathcal{A}$ sends to C what an honest A would have sent to C in execution α_3. For $i \geq 2$, $\mathcal{A}$ examines the message $msg_{i-1}^{\alpha_2}(C,A)_C$. If the message has not been authenticated by B even once, $\mathcal{A}$ authenticates and sends same message to B as

A would have sent to B in round i of execution α_1. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\alpha_2}(C,A)_C$,($\mathcal{A}$ can construct $msg_{i-1}^{\alpha_2}(C,A)_C$, since it passively controls C and has messages received by C in previous round.) such that $msg_{i-1}^{\alpha_2}(C,A)_C \sim msg_{i-1}^{\alpha_1}(C,A)_C$, authenticates it using A's key and sends it to B. If the message has been authenticated by B even once, $\mathcal{A}$ simply authenticates $msg_{i-1}^{\alpha_2}(C,A)_C$ using A's key and sends it to B. Similarly $\mathcal{A}$ authenticates $msg_{i-1}^{\alpha_2}(B,A)_B$ using A's key and sends it to C.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\alpha_2}(C,A)_C$ and $msg_i^{\alpha_2}(B,A)_B$ via A. (These are round i messages in α_2 sent by C and B respectively to A). Similarly via C, $\mathcal{A}$ obtains messages $msg_i^{\alpha_2}(A,C)_A$ and $msg_i^{\alpha_2}(B,C)_B$ in α_2. (These are also round i messages sent by A and B respectively to C. Players respectively compute these messages according to their input, secret key, protocol run by them and the view they get up to round $i-1$).

Consider execution α from the perspective of B and C. We now show that messages received by B and C in round i of α are same as messages received by B and C respectively in round i of α_2. The central idea is similar to proof of Lemma 20.

Lemma 22 *$msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_2}(x,B)_x$ and $msg_i^{\alpha}(x,C)_x \sim msg_i^{\alpha_2}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$*

Proof: We show that for any round i, adversary can ensure that B receives same messages in α and α_2 i.e. $msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_2}(x,B)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$. Argument for $msg_i^{\alpha}(x,C)_x \sim msg_i^{\alpha_2}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$ follows similarly. We prove the same using induction on height of T_{α}^{B} and $T_{\alpha_2}^{B}$(as shown in Figure 4.8). Note that only nodes present in T_{α}^{B} are A, B, C, A'. Corresponding nodes present in $T_{\alpha_2}^{B}$ are A, B, C, A respectively. Notice that since B' does not appear in T_{α}^{B}, any A' in T_{α}^{B} has an outgoing directed edge only and only to C. Similarly, since C' does not appear in T_{α}^{B}, any A in T_{α}^{B} has an outgoing directed edge only and only to B.

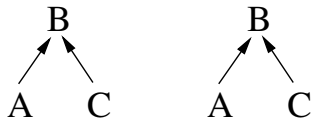

Figure 4.6: T_{α}^{B} and $T_{\alpha_2}^{B}$ at the end of round 1.

We begin analyzing the executions trees T_{α}^{B} and $T_{\alpha_2}^{B}$ in a bottom up manner. Consider trees T_{α}^{B} and $T_{\alpha_2}^{B}$ at the end of round 1 as shown in Figure 4.3. C starts with same input, secret key and executes same code in α and α_2. Thus it will send same messages to B in round 1 of α and α_2 i.e. $msg_1^{\alpha}(C,B)_C \sim msg_1^{\alpha_2}(C,B)_C$. Since A is faulty in α_2, $\mathcal{A}$ can ensure that $msg_1^{\alpha}(A,B)_B \sim msg_1^{\alpha_2}(A,B)_B$. Thus B gets same messages at the end of round 1 in α and α_2 i.e. $msg_1^{\alpha}(x,B)_x \sim msg_1^{\alpha_2}(x,B)_x$, $\forall x \in \mathbb{P}$.

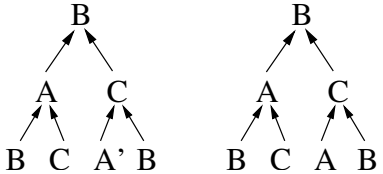

Figure 4.7: T_{α}^{B} and $T_{\alpha_2}^{B}$ at the end of round 2.

We now claim that the similarity holds in round 2 as well i.e. $msg_2^{\alpha}(x,B)_x \sim msg_2^{\alpha_2}(x,B)_x$. Consider trees T_{α}^{B} and $T_{\alpha_2}^{B}$ at the end of round 2 as shown in Figure 4.7. B as well as C start with same input value, secret key and execute same code in both α and α_2 respectively, thus $msg_1^{\alpha}(B,A)_B \sim msg_1^{\alpha_2}(B,A)_B$, $msg_1^{\alpha}(C,A)_C \sim msg_1^{\alpha_2}(C,A)_C$ and $msg_1^{\alpha}(B,C)_B \sim msg_1^{\alpha_2}(B,C)_B$. $\mathcal{A}$ can ensure that $msg_1^{\alpha}(A',C)_{A'} \sim msg_1^{\alpha_2}(A,C)_A$. At the end of round 1, A receives same messages in α and α_2 and has same input value, secret key and executes same code, thus $msg_2^{\alpha}(A,B)_A \sim msg_2^{\alpha_2}(A,B)_A$. Similarly, since C also receives same messages in α and α_2 and has same input value, secret key and executes same code, thus $msg_2^{\alpha}(C,B)_C \sim msg_2^{\alpha_2}(C,B)_C$. Thus, $msg_2^{\alpha}(x,B)_x \sim msg_2^{\alpha_2}(x,B)_x$, $\forall x \in \mathbb{P}$.

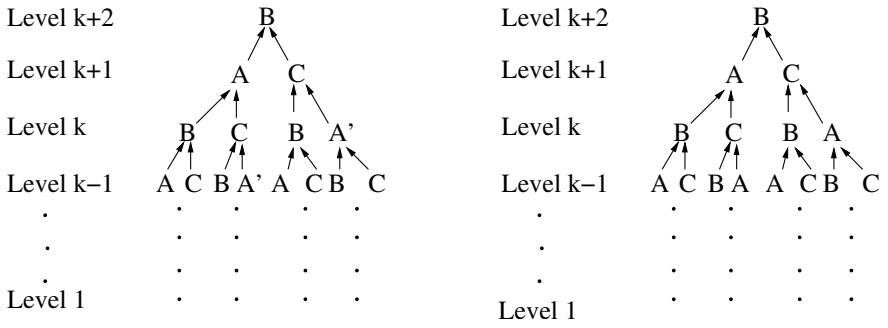

Figure 4.8: T_{α}^{B} and $T_{\alpha_2}^{B}$ at the end of $k+1$ rounds.

Let the similarity be true till some round k i.e. $msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_2}(x,B)_x$, $\forall i | 1 \leq i \leq k$, $\forall x \in \mathbb{P}$. We now show that $\mathcal{A}$ can ensure that the similarity holds for round $k+1$ also. Consider T_{α}^{B} and $T_{\alpha_2}^{B}$ at the end of $k+1$ rounds as shown in Figure 4.8. For proving the induction step, we need to show that B at level $k+2$ receives same messages in both trees. Consider edges between level k and $k+1$. From induction hypothesis any node B up to level k receives same messages in T_{α}^{B} and $T_{\alpha_2}^{B}$. Since B starts with same input value, secret key and executes same code in both α and α_2 respectively, thus will send same messages in round k i.e. $msg_k^{\alpha}(B,A)_B \sim msg_k^{\alpha_2}(B,A)_B$ and $msg_k^{\alpha}(B,C)_B \sim msg_k^{\alpha_2}(B,C)_B$. Similarly one can argue that $msg_k^{\alpha}(C,A)_C \sim msg_k^{\alpha_2}(C,A)_C$. This is because from the induction hypothesis step on heights of T_{α}^{C} and $T_{\alpha_2}^{C}$, one gets $msg_i^{\alpha}(x,C)_x \sim msg_i^{\alpha_2}(x,C)_x$, $\forall i | 1 \leq i \leq k$, $\forall x \in \mathbb{P}$. Now consider A' at level k in in T_{α}^{B} and corresponding A at level k in in $T_{\alpha_2}^{B}$. For time being assume A' up to level k in T_{α}^{B} receives same messages as corresponding A in $T_{\alpha_2}^{B}$. Since A is corrupt in α_2, $\mathcal{A}$ can always ensure that in round k of α_2, A sends to C what A' sends to C in round k of α. Thus A at level $k+1$ receives same messages in both T_{α}^{B} and $T_{\alpha_2}^{B}$. Since A starts with same input value, secret key and executes same code in both α and α_2 respectively, one gets $msg_{k+1}^{\alpha}(A,B)_A \sim msg_{k+1}^{\alpha_2}(A,B)_A$. Similarly, one gets $msg_{k+1}^{\alpha}(C,B)_C \sim msg_{k+1}^{\alpha_2}(C,B)_C$. Thus induction holds for round $k+1$ too. The proof is based on a assumption that A' at level k in T_{α}^{B} receives same messages as corresponding A in $T_{\alpha_2}^{B}$. Note that A' in T_{α}^{B} and A in $T_{\alpha_2}^{B}$ receives messages from B and C. Using induction and arguments similar to those given above one can show that such an assumption indeed holds true. Thus $msg_i^{\alpha}(x,B)_x \sim msg_i^{\alpha_2}(x,B)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$ holds true. ∎

Lemma 23 *$view_B^{\alpha} \sim view_B^{\alpha_2}$ and $view_C^{\alpha} \sim view_C^{\alpha_2}$*

Proof: Follows from equation 4.3 and Lemma 22. ∎

Adversary for α_3:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. For round 1, $\mathcal{A}$ sends to C what an honest B would have sent to C in α_2 and $\mathcal{A}$ sends to A what an honest B would have sent to A in α_2. For $i \geq 2$, $\mathcal{A}$ authenticates $msg_{i-1}^{\alpha_3}(C,B)_C$ using B's key and sends it to A. For $msg_{i-1}^{\alpha_3}(A,B)_A$, $\mathcal{A}$ examines the message. If the message has not been authenticated by C even once, then $\mathcal{A}$ authenticates and sends same message to C as an honest B would have sent to C in round i of execution α_2. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\alpha_3}(A,B)_A$,($\mathcal{A}$ can construct $msg_{i-1}^{\alpha_3}(A,B)_A$, since it passively controls A and has messages received by A in previous rounds.) such that $msg_{i-1}^{\alpha_3}(A,B)_A \sim msg_{i-1}^{\alpha_2}(A,B)_A$, authenticates it using B's key and sends it to C. If the message has been authenticated by C even once, $\mathcal{A}$ simply authenticates $msg_{i-1}^{\alpha_3}(A,B)_A$ using B's key and sends it to C.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\alpha_3}(A,B)_A$ and $msg_i^{\alpha_3}(C,B)_C$ in α_3 via B. (These are round i messages sent by A and C respectively to B). Similarly via A, $\mathcal{A}$ obtains messages $msg_i^{\alpha_3}(B,A)_B$ and $msg_i^{\alpha_1}(C,A)_C$ in α_3. (These are also round i messages sent by B and C respectively to A. Players respectively compute these messages according to their input, secret key, protocol run by them and the view they get up to round $i-1$).

Owing to symmetry of system L, using the proof technique similar to one used in proof of Lemma 20, 21, one can prove the following:

Lemma 24 *$msg_i^{\alpha}(x,C)_x \sim msg_i^{\alpha_3}(x,C)_x$ and $msg_i^{\alpha}(x,A')_x \sim msg_i^{\alpha_3}(x,A)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$*

Lemma 25 *$view_C^{\alpha} \sim view_C^{\alpha_3}$ and $view_{A'}^{\alpha} \sim view_A^{\alpha_3}$.*

As an interesting observation, it appears that the proof of Lemma 20, 22, 24 requires *directed* system, unlike undirected systems used in extant literature [FLM85, LLR02].

4.6 Complete Characterization

We now give the necessary and sufficient conditions for ABG_{mix} tolerating a (t_b,t_p)-adversary over any completely connected synchronous network. As a prelude we first show that there does not exist any protocol solving ABG_{mix} over a complete network $\mathcal{N}'$(Figure 4.9) of four nodes, tolerating adversary basis $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. For the rest of this chapter, $((x_1,\ldots,x_i),(y_1,\ldots,y_j))$ represents a single element of adversary basis such that adversary can corrupt $x_1,\ldots,x_i$ actively and simultaneously control $y_1,\ldots,y_j$ passively. The proof technique is similar to one used in proof of Lemma 18.

Lemma 26 *There does not exist any protocol solving ABG_{mix} over a complete network $\mathcal{N}'$ of four nodes $\mathbb{P} = \{A,B,C,D\}$, tolerating adversary basis $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$.*

Proof: We begin by assuming that there exists a protocol η that solves ABG_{mix} over a completely connected network $\mathcal{N}'$ of four nodes, tolerating a adversary basis $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. The proof proceeds to show that there exists an input assignment where the real process adversary $\mathcal{A}$ (characterized by $\mathbb{A}$) can ensure that non-faulty players do not have a consistent output. In contrast, in the corresponding ideal execution all the non-faulty 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 similar, thus violating Definition 10.

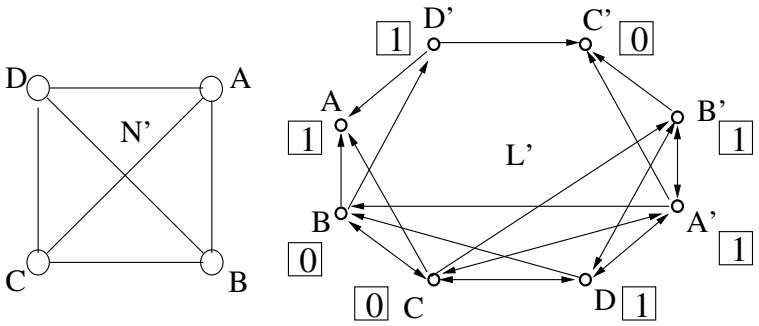

Figure 4.9: Network $\mathcal{N}'$ and System L'.

Using η we create a protocol η'[Definition 12] in such a way that if η exists then so does η'(Lemma 27). Using two copies of η' we construct a system L' (as shown in Figure 4.9), and show that L' must exhibit contradictory behaviour. It follows that our assumption about existence of η is wrong.

We do not know what system L' solves. Formally, system L' is a synchronous system with a well defined behaviour. That is, L' has a well defined output distribution for any particular input assignment. We show that for a particular input assignment, no such well defined behaviour is possible. Further, no player in L' knows the complete system. Each player in aware of only his immediate neighbours. In reality a player may be connected to either a or a', but it cannot differentiate between the two. It knows its neighbour only by its local name which may be a. Specifically, L' is constructed in a such a way that whatever messages are sent to some selected players in L', adversary can ensure same messages to those very selected players in $\mathcal{N}'$. Further, in-neighbourhood of any node a(or a') in L' is same as in-neighbourhood of corresponding node a in $\mathcal{N}'$.

Let β_1, β_2 and β_3 be three scenarios in execution of η over $\mathcal{N}'$. In β_1, B is the General starting with input 0. Adversary $\mathcal{A}$ corrupts A,D actively and controls B passively. In β_2, B is the General. $\mathcal{A}$ corrupts B and interacts with C as if BA started with input 0 and interact with A, D as if B started with input 1. Further, $\mathcal{A}$ controls A passively. In scenario β_3, B is the General starting with input 1. $\mathcal{A}$ corrupts C actively and controls B passively. Further, let β be an execution of L' where each player starts with input value as shown in Figure 4.9. All the players in β are honest and follow the designated protocol correctly.

We now claim that no matter for how many rounds η executes, for any round i, $\mathcal{A}$ can ensure that whatever view [equation 4.2] B,C get in β, $\mathcal{A}$ can generate the same view for B,C in β_1. Similarly we prove that whatever view C,D,A' get in β, $\mathcal{A}$ can generate the same view for C,D,A respectively in β_2. Similarly, whatever view A',B',D get in β, $\mathcal{A}$ can generate the same view for A,B,D respectively in β_3. We prove our claims in Lemma 28 to Lemma 33.

From the definition of ABG_{mix} [Definition 10], in β_1, both B, C should decide on value 0. Since view of B, C is same in β_1 and β, both A, B in β will also decide on value 0 (We are able to make claims regarding the outputs of B and C in β as their views are same as those in β_1. Thus by analyzing their outputs in β_1, we can determine there outputs in β.). Similarly, $\mathcal{A}$ can ensure that view A', B', D in *beta* is same as view of A, B, D in β_3. A, B, D in β_3 will eventually decide upon value 1. Then so should A', B', D in *beta*. Now C, D have same view in β and β_2. As per Definition 10 C, D in β_2 should agree on same value. Then so should C, D in β. But C,D have already decided upon values 0 and 1 respectively in β. This implies L' must exhibit contradictory behaviour. ∎

To complete the above proof we show that – B, C get same view in β and β_1; C, D get same view in β and β_2 and view of A', B', D in *beta* is same as view of A, B, D respectively in β_3. We prove the same in Lemmas 28 - 33. As a prelude, we define the protocol η'[Definition 12] and show that if η exists then so does η'(Lemma 27). ∎

Definition 12 (η') *All statements in η' are same as those in η except the following:*

- *any statement in η of the kind "A sends message m to B" is replaced by "A multicasts message m to all instances of B"(i.e. B,B') [6]. Similarly, "A sends message m to C" is replaced by "A multicasts message m to all instances of C"(i.e. C,C').*
- *any statement in η of the kind "B sends message m to D" is replaced by "B multicasts message m to all instances of D"(i.e. D,D').*
- *any statement in η of the kind "C sends message m to A" is replaced by "C multicasts message m to all instances of A"(i.e. A,A'). Similarly, "C sends message m to B" is replaced by "C multicasts message m to all instances of B"(i.e. B,B').*
- *any statement in η of the kind "D sends message m to B" is replaced by "D multicasts message m to all instances of B"(i.e. B,B').*

Lemma 27 *If η exists then η' exists.*

Proof: Implied from Definition 12. ∎

For scenario β_1, we now specify the behaviour of the adversary:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. In round 1, $\mathcal{A}$ sends to C what an honest A and D would have sent to C in round 1 of β_2. For $i \geq 2$, $\mathcal{A}$ authenticates $msg_{i-1}^{\beta_1}(C, A)_C$ using A's secret key and sends it to B,D. Similarly, $\mathcal{A}$ authenticates $msg_{i-1}^{\beta_1}(C, D)_C$ using D's secret key and sends it to A,B. For $msg_{i-1}^{\beta_1}(B, A)_B$, $\mathcal{A}$ examines the message. If the message has not been authenticated by C even once then $\mathcal{A}$ authenticates and sends same message to C as an honest A would have sent to C in β_2. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\beta_1}(B, A)_B$, such that $msg_{i-1}^{\beta_1}(B, A)_B \sim msg_{i-1}^{\beta_2}(B, A)_B$, authenticates it using A's key and sends it to C. If $msg_{i-1}^{\beta_1}(B, A)_B$ has been authenticated by C even once, $\mathcal{A}$ simply authenticates the message using A's key and sends it to C. Likewise $\mathcal{A}$ examines $msg_{i-1}^{\beta_1}(B, D)_B$. If the message has not been authenticated by C even once $\mathcal{A}$ authenticates and sends same message to C as an honest D would have sent to C in execution β_2. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\beta_1}(B, D)_B$ such that $msg_{i-1}^{\beta_1}(B, D)_B \sim msg_{i-1}^{\beta_2}(B, D)_B$, authenticates it using D's key and sends it to C. If $msg_{i-1}^{\beta_1}(B, D)_B$ has been authenticated by C even once, $\mathcal{A}$ authenticates the message using D's key and sends it to C.
2. *Receive incoming messages of round i:* $\mathcal{A}$ obtain messages $msg_i^{\beta_1}(B, A)_A$, $msg_i^{\beta_1}(C, A)_C$ and $msg_i^{\beta_1}(D, A)_D$ via A. Similarly via D $\mathcal{A}$ gets $msg_i^{\beta_1}(A, D)_A$, $msg_i^{\beta_1}(B, D)_B$ and $msg_i^{\beta_1}(C, D)_C$. (These are round i messages sent by B,C, D to A and A,B,C to D respectively). Similarly, $\mathcal{A}$

[6] B and B' are independent copies of B with same authentication key.

obtains $msg_i^{\beta_1}(A,B)_A$, $msg_i^{\beta_1}(C,B)_C$ and $msg_i^{\beta_1}(D,B)_D$ via B. (These are round i messages sent by A,C,D to B. A,C,D respectively compute these messages according to their input value, secret key, protocol run by them and the view they get up to receive phase of round $i-1$.)

We now show that the messages received by B,C in round i of β are same as the messages received by B,C respectively in round i of β_1. Our technique is same as one used in proof of Lemma 20

Lemma 28 *$msg_i^{\beta}(x,B)_x \sim msg_i^{\beta_1}(x,B)_x$ and $msg_i^{\beta}(x,C)_x \sim msg_i^{\beta_1}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.*

Proof: We show that for any round i, adversary can ensure that B receives same messages in β and β_1 i.e. $msg_i^{\beta}(x,B)_x \sim msg_i^{\beta_1}(x,B)_x$, $\forall x \in \mathbb{P}$. Argument for $msg_i^{\beta}(x,C)_x \sim msg_i^{\beta_1}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$ follows similarly. We apply induction on heights of T_{β}^B and $T_{\beta_1}^B$ (as shown in Figure 4.12). Note that only nodes present in T_{β}^B are B, C, D, A', B'. Corresponding nodes present in $T_{\beta_1}^B$ are B, C, D, A, B respectively.

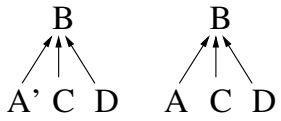

Figure 4.10: Execution trees T_{β}^B and $T_{\beta_1}^B$ at the end of round 1.

We analyze these trees in bottom up manner. Consider trees T_{β}^B and $T_{\beta_1}^B$ at the end of round 1 as shown in Figure 4.10. C starts with same input, secret key and executes same code in β and β_1. Thus it will send same messages to B in round 1 of β and β_1 i.e. $msg_1^{\beta}(C,B)_C \sim msg_1^{\beta_1}(C,B)_C$. Since A and D are faulty in β_1, aforementioned adversary $\mathcal{A}$ can ensure that $msg_1^{\beta}(A',B)_{A'} \sim msg_1^{\beta_1}(A,B)_A$ and $msg_1^{\beta}(D,B)_D \sim msg_1^{\beta_1}(D,B)_D$. Thus B gets same messages at the end of round 1 in β and β_1 i.e. $msg_1^{\beta}(x,B)_x \sim msg_1^{\beta_1}(x,B)_x$, $\forall x \in \mathbb{P}$.

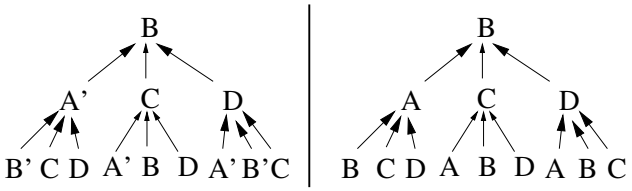

Figure 4.11: Execution trees T_{β}^B and $T_{\beta_1}^B$ at the end of round 2.

We now claim that the similarity holds for round 2 as well i.e. $msg_2^{\beta}(x,B)_x \sim msg_2^{\beta_1}(x,B)_x$, $\forall x \in \mathbb{P}$. Consider trees T_{β}^B and $T_{\beta_1}^B$ at the end of round 2 as shown in Figure 4.11. Consider node B at level 1 in T_{β}^B and $T_{\beta_1}^B$. Node B starts with same input value, secret key and execute same code in both β and β_1 respectively, thus $msg_1^{\beta}(B,C)_B \sim msg_1^{\beta_1}(B,C)_B$. Since A,D are faulty, $\mathcal{A}$ can ensure that $msg_1^{\beta_1}(A,C)_A \sim msg_1^{\beta}(A',C)_{A'}$ and $msg_1^{\beta_1}(D,C)_D \sim msg_1^{\beta}(D,C)_D$. Thus C receives same messages at the end of round 1 in β and β_1. Since C starts with same input value, secret key and execute same code in both β and β_1 respectively, it sends same message to B in round 2 i.e. $msg_2^{\beta}(C,B)_C \sim msg_2^{\beta_1}(C,B)_C$. Now consider A' at level 2 in T_{β}^B and corresponding A at level 2 in $T_{\beta_1}^B$. B' in β starts with a different input from B in β_1, thus $msg_1^{\beta}(B',A')_{B'} \nsim msg_1^{\beta_1}(B,A)_B$. However since A is faulty and B is passively corrupt in β_1, $\mathcal{A}$ on behalf of B can construct $msg_1^{\beta_1}(B,A)_B$ such that $msg_1^{\beta_1}(B,A)_B \sim msg_1^{\beta}(B',A')_{B'}$. C starts with same input

value, secret key and execute same code in both β and β_1 respectively, thus $msg_1^{\beta}(C, A')_C \sim msg_1^{\beta_1}(C, A)_C$. Since D is faulty, $\mathcal{A}$ can ensure that $msg_1^{\beta_1}(D, A)_D \sim msg_1^{\beta}(D, A')_D$. Thus A' in β receives same messages at the end of round 1 as A in β_1. Since A is faulty in β_1, $\mathcal{A}$ can ensure that A in β_1 sends message to B in round 2 same as what A' in β sends to B in round 2 i.e. $msg_2^{\beta_1}(A, B)_A \sim msg_2^{\beta}(A', B)_{A'}$. Similarly one can show that $mmsg_2^{\beta_1}(D, B)_D \sim msg_2^{\beta}(D, B)_D$. Thus $msg_2^{\beta}(x, B)_x \sim msg_2^{\beta_1}(x, B)_x$, $\forall x \in \mathbb{P}$.

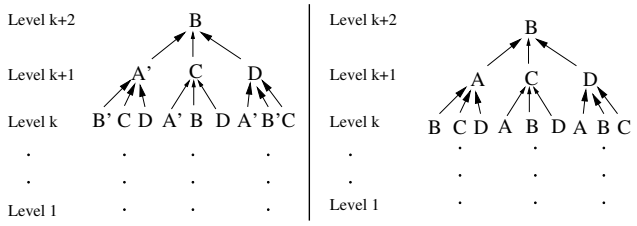

Figure 4.12: T_{β}^{B} and $T_{\beta_1}^{B}$ at the end of $k+1$ rounds.

Let the similarity be true till some round k i.e. $msg_i^{\beta}(x, B)_x \sim msg_i^{\beta_1}(x, B)_x$, $\forall i | 1 \leq i \leq k$, $\forall x \in \mathbb{P}$. We now show that $\mathcal{A}$ can ensure that the similarity holds for round $k+1$ also. Consider T_{β}^{B} and $T_{\beta_1}^{B}$ at the end of $k+1$ rounds as shown in Figure 4.12. To prove the induction step we need to show that B at level $k+2$ receives same messages in both trees. Consider node D at level $k+1$. One can argue that C till round k also gets same messages in β and β_1. This is because from similar induction hypothesis step on heights of T_{β}^{C} and $T_{\beta_1}^{C}$, one gets $msg_i^{\beta}(x, C)_x \sim msg_i^{\beta_1}(x, C)_x$, $\forall i | 1 \leq i \leq k$, $\forall x \in \mathbb{P}$. Now, since C starts with same input value, secret key and execute same code in both β and β_1 respectively, it sends same messages to D in round k i.e. $msg_k^{\beta}(C, D)_C \sim msg_k^{\beta_1}(C, D)_C$. For time being assume A' receives messages till round k in β_1 same as what A receives till round k in β. Since A is faulty in β_1, $\mathcal{A}$ can ensure that A sends same message to D in β_1 as A' sends to D in β i.e. $msg_k^{\beta}(A', D)_{A'} \sim msg_k^{\beta_1}(A, D)_A$. Similarly assume that B' receives messages till round k in β_1 same as what B receives messages till round k in β. But B in β_1 starts with a different input from B' in β, thus they send different messages to D in β and β_1. However since D is faulty and B is passively corrupt in β_1, $\mathcal{A}$ can ensure that $msg_k^{\beta}(B', D)_{B'} \sim msg_k^{\beta_1}(B, D)_B$. Thus D at level $k+1$ receives same messages in T_{α}^{B} and $T_{\alpha_1}^{B}$. Since D is faulty in β_1, $\mathcal{A}$ can ensure that $msg_{k+1}^{\beta}(D, B)_D \sim msg_{k+1}^{\beta_1}(D, B)_D$. Using similar arguments one can show that $msg_{k+1}^{\beta}(C, B)_C \sim msg_{k+1}^{\beta_1}(C, B)_C$ and $msg_{k+1}^{\beta}(A', B)_{A'} \sim msg_{k+1}^{\beta_1}(A, B)_A$. Thus B receives same messages in round $k+1$ of β and β_1. Thus induction hypothesis holds for round $k+1$ too. Thus $msg_i^{\beta}(x, B)_x \sim msg_i^{\beta_1}(x, B)_x, \forall x \in \mathbb{P}$ holds true. The above proof is based on the assumption that A' up to level k in T_{α}^{B} receives same messages as corresponding A in $T_{\beta_1}^{B}$. Using induction and arguments similar to given above one can show easily that both assumptions indeed holds true. Similarly one can prove that B' up to level k in T_{α}^{B} receives same messages as corresponding B in $T_{\beta_1}^{B}$. ∎

Lemma 29 *$view_B^{\beta} \sim view_B^{\beta_1}$ and $view_C^{\beta} \sim view_C^{\beta_1}$*

Proof: Follows from equation 4.3 and Lemma 28. ∎

We now give the adversary for β_2:

1. *Send outgoing messages of round i:* Based on the messages received in round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. In round 1, $\mathcal{A}$ sends to C what an honest B would

have sent to C in round 1 of β_1. Similarly $\mathcal{A}$ sends to D what an honest B would have sent to D in round 1 of β_3 and $\mathcal{A}$ sends to A what an honest B would have sent to A in round 1 of β_3. For $i \geq 2$, $\mathcal{A}$ authenticates $msg_{i-1}^{\beta_2}(C,B)_B$ using B's secret key and sends it to A,D. Similarly, $\mathcal{A}$ authenticates $msg_{i-1}^{\beta_2}(D,B)_D$ using B's secret key and sends it to A,C. For $msg_{i-1}^{\beta_2}(A,B)_A$, $\mathcal{A}$ examines the message. If the message has not been authenticated by either C or D even once, then $\mathcal{A}$ authenticates and sends same message to C as an honest B would have sent to C in β_1. Similarly $\mathcal{A}$ authenticates and sends same message to D as an honest B would have sent to D in β_3. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\beta_2}(A,B)_A$, such that $msg_{i-1}^{\beta_2}(A,B)_A \sim msg_{i-1}^{\beta_1}(A,B)_A$, authenticates it using B's key and sends it to C. Similarly $\mathcal{A}$ constructs $msg_{i-1}^{\beta_2}(A,B)_A$, such that $msg_{i-1}^{\beta_2}(A,B)_A \sim msg_{i-1}^{\beta_3}(A,B)_A$, authenticates it using B's key and sends it to D. If $msg_{i-1}^{\beta_2}(A,B)_A$ has been authenticated by either C or D even once, $\mathcal{A}$ simply authenticates the message using B's key and sends it to C and D.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\beta_2}(A,B)_A$, $msg_i^{\beta_2}(C,B)_C$ and $msg_i^{\beta_2}(D,B)_D$ from B in β_2 (These are round i messages sent by A,C,D to B. They respectively compute these messages according to their input, protocol run by them and the view they get up to receive phase of round $i-1$.). Similarly $\mathcal{A}$ obtains $msg_i^{\beta_2}(B,A)_B$, $msg_i^{\beta_2}(C,A)_C$ and $msg_i^{\beta_2}(D,A)_D$ from A in β_2 (These are round i messages sent by B,C,D to A).

Similar to Lemma 28,29 one can prove the following lemmas.

Lemma 30 *$msg_i^{\beta}(x,C)_x \sim msg_i^{\beta_2}(x,C)_x$, $msg_i^{\beta}(x,D)_x \sim msg_i^{\beta_2}(x,D)_x$ and $msg_i^{\beta}(x,A')_x \sim msg_i^{\beta_2}(x,A)_x$ $\forall i > 0$, $\forall x \in \mathbb{P}$.*

Lemma 31 *$view_C^{\beta} \sim view_C^{\beta_2}$, $view_D^{\beta} \sim view_D^{\beta_2}$, $view_{A'}^{\beta} \sim view_A^{\beta_2}$*

Adversary for β_3:

1. *Send outgoing messages of round i:* Based on the messages received in round $i-1$, $\mathcal{A}$ decides on the messages to be sent in i. In round 1, $\mathcal{A}$ sends to D what an honest C would have sent to D in round 1 of β_2. For $i \geq 2$ $\mathcal{A}$ authenticates $msg_{i-1}^{\beta_3}(A,C)_A$ using secret key of C and sends it to B,D. Similarly it authenticates $msg_{i-1}^{\beta_3}(D,C)_D$ using C's secret key and sends it to A,B. For $msg_{i-1}^{\beta_3}(B,C)_B$, $\mathcal{A}$ examines the message. If the message has not been authenticated by either A or D even once, then $\mathcal{A}$ authenticates and sends same message to A as an honest C would have sent to A in β_2 and sends same to D as an honest C would have sent to D in execution β_2. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{\beta_3}(B,C)_B$, such that $msg_{i-1}^{\beta_3}(B,C)_B \sim msg_{i-1}^{\beta_2}(B,C)_B$ authenticates it using C's key and sends it to A,D. If $msg_{i-1}^{\beta_3}(B,C)_B$ has been authenticated by either of A or D even once, $\mathcal{A}$ simply authenticates the message using C's key and sends it to A,D.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{\beta_3}(A,C)_A$, $msg_i^{\beta_3}(B,C)_B$ and $msg_i^{\beta_3}(D,C)_D$ via C. (These are round i messages sent by A,B and D to C). Similarly $\mathcal{A}$ obtains $msg_i^{\beta_3}(A,B)_A$, $msg_i^{\beta_3}(C,B)_C$ and $msg_i^{\beta_3}(D,B)_D$ via B. (These are round i messages sent by A,C and D to B. A,C and D respectively compute these messages according to the protocol run by them and the view they get receive phase of round $i-1$.)

Similar to Lemma 28,29 one can prove the following lemmas.

Lemma 32 *$msg_i^{\beta}(x, A')_x \sim msg_i^{\beta_3}(x, A)_x$, $msg_i^{\beta}(x, B')_x \sim msg_i^{\beta_3}(x, B)_x$, and $msg_i^{\beta}(x, D)_x \sim msg_i^{\beta_3}(x, D)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.*

Lemma 33 *$view_{A'}^{\beta} \sim view_A^{\beta_3}$, $view_{B'}^{\beta} \sim view_B^{\beta_3}$, $view_D^{\beta} \sim view_D^{\beta_3}$.*

We now present the main theorem of this chapter.

Theorem 34 (Main Theorem) *ABG_{mix} over a completely connected synchronous network $\mathcal{N}'$ of n nodes tolerating (t_b,t_p)-adversary is possible if and only if if $n > 2t_b + min(t_b, t_p)$, for $t_p > 0$.*

Proof: We first give necessity proof followed by proof for sufficiency.

`Necessity:` We prove that there does not exist any protocol solving ABG_{mix} over a completely connected synchronous network $\mathcal{N}'$ of n nodes tolerating (t_b,t_p)-adversary when $n \leq 2t_b + min(t_b, t_p)$, for $t_p > 0$. We present the proof separately for $t_b > t_p$ and $t_b \leq t_p$.

`Case of` $t_b > t_p$`:` We assume that there exists a protocol η solving ABG_{mix} complete network $\mathcal{N}'$ tolerating (t_b,t_p)-adversary when $n \leq 2t_b + min(t_b, t_p)$. Using η, we construct a protocol η' which solves ABG_{mix} over a completely connected graph of four nodes, tolerating $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. However, from Lemma 26, we know that there cannot exist any such η'. This contradicts our assumption that there exists a solution η solving ABG_{mix} for $n \leq 2t_b + min(t_b, t_p)$, $t_p > 0$.

We now show as to how to transform η into a solution η' which solves ABG_{mix} over a completely connected graph of four nodes, tolerating $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. Divide n players in η into sets I_A, I_B, I_C, I_D, such that their respective sizes are $min(t_b, t_p), min(t_b, t_p), t_b$, $(t_b - min(t_b, t_p))$. $\mathbb{A}$ can corrupt all the players in any of the following sets $I_A, I_B, I_C, I_D, (I_A \cup I_D), (I_B \cup I_D)$ actively and players in I_A, I_B, I_D passively. Note that the players from the set I_C cannot be corrupted passively. Each of the four players A,B,C and D in η' simulate players in I_A, I_B, I_C, I_D respectively. Each player i in η' 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 value v, then so does player i.

We now prove that if η solving ABG_{mix} tolerating (t_b,t_p)-adversary when $n \leq 2t_b + min(t_b, t_p)$, then η' solves ABG_{mix} tolerating $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. For simplicity we assign any actively and passively corrupted players of η to be exactly those that are simulated by actively and passively corrupted player in η'. Let ψ' be an execution of η' with the faults characterized by $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. Let ψ be an execution of η. As per our assumption ψ solves ABG_{mix}, thus ψ satisfies Definition 10. We now show that same holds for ψ' if it holds for ψ. W.l.o.g in ψ, let the general be from set I_i, then in ψ', player i acts as the general. Note that in ψ if I_i is controlled actively or passively by the adversary, then so is i is ψ'. Let j,k ($j \neq k$) be two non-faulty players in ψ'. j and k simulates at least one player each in ψ. w.l.o.g let them simulate players in I_j, I_k. Since j and kl are non-faulty, so are all players in I_j, I_k. For ψ, all players in I_j, I_k must terminate, then so should j and k. In ψ, all non-faulty players including I_j, I_k should agree on same value say u, then in ψ', j, k also agree on u. In ψ, if the general is non-faulty and starts with value v, then in ψ' too, general will be non-faulty and starts with value v. In such a case in ψ, all non-faulty players including I_j, I_k should have $u = v$, then in ψ', j, k will also have $u = v$. Thus ψ' also satisfies Definition 10. Then, η' solves ABG_{mix} tolerating $\mathbb{A} =$

$\{((A,D),(B)),((B),(A)),((C),(B))\}$.

`Case of` $t_b \leq t_p$: In this case, the expression $n \leq 2t_b + min(t_b, t_p)$ reduces to $n \leq 3t_b$. We assume that there exists a protocol λ solving ABG_{mix} complete network $\mathcal{N}'$ tolerating (t_b,t_p)-adversary when $n \leq 3t_b$. Using λ, we construct a protocol λ' which solves ABG_{mix} over a completely connected graph of three nodes, tolerating a (1,1)-adversary. However, from Theorem 18, we know that there cannot exist any such λ'. This contradicts our assumption that there exists a solution λ solving ABG_{mix} for $n \leq 3t_b$, when $t_b \leq t_p$.

Given λ, we now show as to how can one construct λ'. Divide the n players into three sets I_1, I_2 and I_3 such each is of size at max t_b i.e. $|I_i| \leq t_b$ (Since $n \leq 3t_b$ such a division is always possible). Further, since $t_b \leq t_p$, it vacuously implies $|I_i| \leq t_p$. Thus, adversary can corrupt any of I_i actively and I_j passively, $i \neq j$. Let player i in λ' simulate all the players in I_i. For simplicity we assign any actively and passively corrupted players in an execution of λ to be exactly those that are simulated by actively and passively corrupted player in corresponding execution of λ'. Similar to the argument presented for $t_b > t_p$, one can show that if λ satisfies Definition 10, then so does λ'. This completes the necessity proof.

`Sufficiency -` For sufficiency we present protocol for $n > 2t_b + \min(t_b, t_p)$, $t_p > 0$. We present the protocol separately for $t_b > t_p$ and $t_b \leq t_p$.

`Case of` $t_b > t_p$: $n > 2t_b + \min(t_b, t_p)$ reduces to $n > 2t_b + t_p$. For this we present a protocol and prove its correctness in section 4.6.1.

`Case of` $t_b \leq t_p$: $n > 2t_b + \min(t_b, t_p)$ reduces to $n > 3t_b$. Here any protocol for unauthenticated Byzantine Generals Problem works (such as *EIGByz* protocol given in section 3.4.2). This is because for unauthenticated setting $t_p = n - t_b$. This completes the sufficiency proof. We remark that for t_p=0, the result reduces to $n > t_b$ [PSL80]. ∎

4.6.1 Protocol for $n > 2t_b + t_p$

The proposed protocol is obtained by a sequence of transformations on EIG tree [BNDDS87]. A detailed description of the construction of EIG tree is available in section 3.4.2. Our protocol *EIGPrune* is given in Figure 4.13.

***EIGPrune* Algorithm**

General $\mathcal{G}$ send his value to every player. Every player assumes this value from the $\mathcal{G}$ as his input value and and exchanges messages with others as per *EIGStop* protocol in section 3.4.2 for $t_b + t_p + 1$ rounds.

At the end of $t_b + t_p + 1$ rounds of *EIGStop* protocol, player p_i invokes **Prune**(EIG) [Definition 13]. Player p_i applies the following decision rule – take majority of the values at the first level [7] of its *EIG* tree (note that he does not need to take a majority over the entire *EIG* tree). If a majority exists player, p_i decides on that value; else, p_i decides on *default value*, v_0.

Figure 4.13: *EIGPrune* algorithm

Definition 13 (Prune(EIG)) *This method that takes an EIG tree as an input and deletes 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 as given in the sequel. 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 appears in* $subtree_j{}^i$*. If* $|W_j| > 1$*,* $subtree_j{}^i$ *is deleted and modified EIG tree is returned.*

We prove the correctness of *EIGPrune* via Lemma 35 – 38.

Lemma 35 *The* $subtree_j{}^i$*, where* j *is an honest player and* i *is a non-faulty player, will never be deleted during* **Prune**(EIG) *operation.*

Proof: This Lemma stems from the fact that any message signed by an honest player cannot be changed in the course of the protocol. Thus, a $subtree_j{}^i$, j being an honest player will never be deleted in **Prune**(*EIG*) and will be consistent throughout for all non-faulty players. ∎

Lemma 36 *After* $t_b + t_p + 1$ *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 non-faulty.*

Proof: Any message sent in $(t_b + t_p)^{th}$ round has a label of length $t_b + t_p$ and hence we are sure to have either an honest player already having signed on it or in $(t_b + t_p + 1)^{th}$ round an honest player would broadcast it. This ensures that a value cannot be changed/reintroduced in the $(t_b + t_p + 1)^{th}$ round. In other words, a faulty player can either send different initial values in round one or change a value in Round k, $2 \leq k \leq t_b + t_p$, if and only if all players who have signed so far on that message are under the control of adversary. In any case, the non-faulty players send these values in the next round and hence the Lemma. ∎

Lemma 37 $subtree_j{}^i$ *and* $subtree_j{}^k$ *in the* EIG *trees of any two players* i, k *will have same values after the subjecting the tree to* **Prune**(EIG)*, where* i, k *are non-faulty players.*

Proof: This follows from previous Lemma 36 as, if subtrees had different values; then as per the protocol they would have broadcasted the values in their *EIG* tree in the next round and thus the subtrees would have more than one different value resulting in their deletion during **Prune**(*EIG*) step. ∎

Lemma 38 *For* $n > 2t_b + t_p$*,* EIGPrune *algorithm solves* ABG_{mix}*.*

Proof: $n - (t_b + t_p)$ represents the number of honest players and according to $n > 2t_b + t_p$, $n - (t_b + t_p) > t_b$. Thus honest majority is guaranteed which vacuously implies non-faulty majority. The decision rule ensures that in case the General is non-faulty and starts with v, all non-faulty players decide on v. Further if the General is faulty, all non-faulty should agree on same value. Let i and j be any two non-faulty players. Since, decisions only occur at the end, and by previous lemma we see that $\forall i, subtree_j{}^i$ can have only one value which consistent throughout all $subtree_j^i, \forall i \in \mathbb{P}$. This implies they have the same set of values. The decision rule then simply implies that i and j make the same decision. ∎

Chapter 5

On Composition of ABG

5.1 Introduction

For a large part of the literature on security, it is common to assume that a protocol is executed in "isolation" i.e. when an instance of the given protocol is executed, no instance of any other protocol (including currently executed protocol) is in execution concurrently. This is popularly referred as "stand alone" execution model. However, over most real life scenarios, a protocol is seldom executed in a stand alone setting. In most networks that we come across in our day-to-day life, many different protocols are run simultaneously. Can a protocol proven to be secure in stand alone setting, become insecure when run in presence of other (proven to be secure) protocols ?

The answer to the above question can be (surprisingly) *yes*. This is because, adversary *may be able* to disrupt the given protocol using additional information which it may gain from the "environment" consisting of other concurrent executions. Some of the first work in this line was on the problem of zero-knowledge and concurrent zero-knowledge [CKPR01, DNS04, GK96]. Informally, it can be said that: *secure + secure is not always secure* i.e. protocols which are proven to be secure in stand alone settings can become insecure when run concurrently in presence of other secure protocols [NY90, DDN91, RS92, CIO98, FF00, GK96, DNS04]. In many complex settings where a secure protocol is composed with an arbitrary set of secure protocols, or more generally when the protocol is used as a component of an arbitrary system how does one guarantee non-malleability of secure protocols ? One approach could be to model the all the protocols executing concurrently and include them in the definition of security of the given protocol [DNS04, GM00]. However, this approach not only makes the definition of security of a given protocol very cumbersome but altogether fails when the environment is *not known* a priori. For complex networks such as Internet it will be unduly optimistic to assume that the protocol designer has a priori knowledge of which all protocols can ever be executed in the environment. In order to circumvent this problem, Canetti [Can01a] took an alternative approach. He introduced the notion of *Universal Composability* to study the implications on security of protocols when run in arbitrary any unknown environment. The central idea is to use the stand alone definition for security of any protocol and show that the definition is satisfied for any arbitrary environment. Some of the subsequent papers in this line are [CR03, PS04, CK02, CF01, CLOS02, Lin03b, PR03, Lin03c].

In continuation, Lindell *et al.* [LLR02, LLR06] introduced the problem of ABG under parallel composition. They proved that for $n > t$, protocols for ABG fail to remain secure even when two instances of the same protocol are executed in parallel. Subsequently, Lindell *et al.* proved a stronger result that if $n \leq 3t$, there *cannot* exist any protocol solving ABG that composes in parallel even twice (assuming no joint state). The impossibility arises due to ability of the adversary to

fail any protocol by borrowing messages from other execution. 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 $n > 3t$. The result essentially implies that under parallel composition, power of authentication is rendered useless. However on a more optimistic note, they show that if each run of the protocol is further augmented with *unique session identifiers*, one can design ABG protocols that compose in parallel for any number of executions, tolerating $t < n$ faults.

5.1.1 Our Pursuit

In this chapter, we study the problem of ABG under parallel composition. We argue that the results in the state-of-the-art implicitly assumes the following: *if the adversary corrupts(in Byzantine fashion) a player in any one of the parallel executions of the ABG protocol, then the adversary will also corrupt(in Byzantine fashion) the same very player in every other parallel execution(of the given ABG protocol) too.* Clearly, this is a very strong presumption with respect to adversary's behaviour. The rationale being: with respect to stand alone execution of protocols, it is standard to assume that a t-adversary may not always use his full power to thwart a protocol and may as well corrupt less than t players if it helps the adversary. With respect to parallel composition of protocols, a t-adversary can choose to use lesser power in the following ways – (1) the adversary may corrupt some players (less than t in number) across all executions, (2) adversary may corrupt different players in different executions such that the total number of players corrupt in at least one execution is $\leq t$, (3) any combination of 1 and 2. Note that such an adversary is a valid t-adversary.

Motivated from this we study the consequences of removing the aforementioned assumption on the problem of ABG under parallel composition. In the absence of the aforementioned assumption we prove that, for $n < 2t$, there *does not* exist any ABG protocol, that composes in parallel even twice, in spite of using unique session identifiers. Further, for $n \geq 2t$, we design ABG protocols that compose for any number of parallel executions(when supplemented with unique session identifiers).

5.1.2 Protocol Composition

Generally speaking, the notion of protocol composition refers to a setting where many protocol executions take place. This includes many possible scenarios, ranging from the case where a single set of players run the same protocol many times to the case that many sets of different players run many different protocols many times.

A standard notion of composition is that of *stateless* composition. This means that the honest parties relate to 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 keep track of the history of past executions. (This is called stateless composition because no joint stateis kept between the executions.) One prefers stateless composition as the complexity of simultaneously coordinating between many executions can be high, and keeping a lot of state information can become a burden on the system. However, even if the amount of joint state required is manageable, stateful composition may still be difficult, if not impossible, to carry out. For example, ensuring successful coordination between protocols that are designed independently of each other may be very problematic.

Note that in contrast to the honest parties, the adversary may keep joint state and coordinate its actions between the protocol executions. This asymmetry between the adversary (who is stateful) and the honest parties (who are stateless), is due to the fact that some level of coordination is

clearly possible. Thus, although it is undesirable to rely on such coordination in the construction of protocols, it would be careless to assume that the adversary cannot utilize it to some extent. Furthermore, the adversary selects its strategy after all protocols are designed and can therefore implement a (joint) malicious strategy that takes all protocol executions into account.

Types of Protocol Composition

There are three important parameters based on which protocol composition is characterized, namely: the *context* in which the protocol runs, the *participating players* and the *scheduling*.

1. **The context:** This refers to the question of *which protocols* are being run together in the network, or in other words, with which protocols should the protocol in question compose. There are two contexts that have been considered, defining two classes of composition:
   (a) Self composition: A protocol is said to *self compose* if it remains secure when it alone is executed many times in a network. We stress that in this setting, there is only one protocol that is being run.
   (b) General Composition: Here many different protocols are run together in the network. Furthermore, these protocols may have been designed independently of one another. A protocol is said to remain secure under *general composition* if its security is maintained even when it is run along with other arbitrary protocols.
2. **The participating players:** This addresses whether or not the same set of players is involved in all executions:
   (a) A single set of players: In this setting, same set of players participates in all executions.
   (b) Arbitrary sets of players: In this setting, arbitrary (and possibly intersecting) sets of parties run each protocol execution.
3. **The scheduling:** Literature considers three main types of scheduling:
   (a) Sequential: Each new execution begins strictly after the previous one terminates. Thus, at any give time, only one protocol is running.
   (b) Parallel: All executions begin at the same time and proceed at the same rate (i.e., in a synchronous fashion).
   (c) Concurrent: The scheduling of the protocol executions, including when they start and the rate at which they proceed, is determined by the adversary. That is, the adversary has full control over when messages sent by the parties are delivered (as is typical in an asynchronous network).

Throughout this chapter we work with self composition, single set of players and parallel composition.

5.2 Our Model

Consider a set of n players $\mathbb{P}=\{p_1, p_2 \dots p_n\}$ 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. During the execution of the protocol, the adversary 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 "magical"[1] (signature/authentication) scheme via which players authenticate themselves. 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).

5.3 Ramifications of Removing the "Assumption"

Is $n > t$ sufficient for parallel composition of ABG protocols(with unique session identifiers) if the adversary can corrupt different players in different executions ? We answer this question in negative. This implies that the results in the state-of-the-art on composition of ABG protocols implicitly assume that if the adversary corrupts a player in any one of the parallel executions of the ABG protocol, then the adversary will also corrupts the same very player in every other parallel execution too. In section 5.8, we show that in absence of such an assumption, the proof for sufficiency of $n > t$ given in the extent literature breaks down.

In order to understand our results, it is important to understand the *implications* of permitting the adversary to corrupt different players in different executions. We do the same with the help of a simple scenario: Consider a set of n players running two parallel executions, say E_1 and E_2, of some ABG protocol(proven to be secure in stand alone setting). Let P_a be one such player among the set of n players. Adversary corrupts(in Byzantine fashion) P_a only in E_1. The two executions E_1 and E_2 executed by P_a can also be perceived as processor P_a running two threads E_1 and E_2 in parallel, as shown in Figure 5.1.

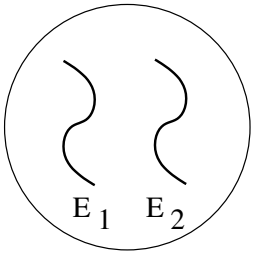

Figure 5.1: Snap shot of memory of processor P_a running two threads, E_1 and E_2, in parallel.

Further, assume that P_a authenticates its messages in E_1 and E_2 using distinct authentication keys, say k_1 and k_2 respectively. We claim that by virtue of corrupting P_a in E_1, adversary can forge messages on behalf of P_a in E_2 even though P_a is honest in E_2.

Claim 39 *By corrupting P_a in Byzantine fashion only in E_1, adversary can forge messages on behalf of P_a in E_2 even though P_a is honest in E_2.*

Proof: In order to forge messages on behalf of P_a in E_2, adversary needs the key used by P_a in execution E_2. Our claim stems from the following observation: Since P_a is Byzantine corrupt in E_1, E_1 can be treated as a faulty thread. Thus, adversary can always execute that code in E_1, which

[1]Refer to assumption 1, section 2.2

can **pull** the private key k_2 (or for that matter any data internal to E_2) from thread E_2(Figure 5.2). Thus, adversary can always read the key used by P_a in execution E_2, hence the claim. ∎

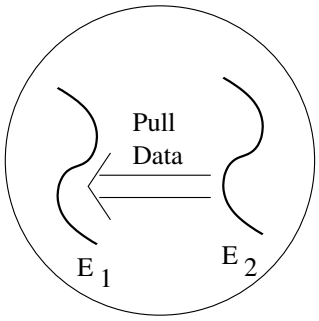

Figure 5.2: Snap shot of memory of processor P_a running two threads, E_1 and E_2, in parallel. Even though E_2 is non-faulty thread, since E_1 is a faulty thread, adversary can always read any internal data of E_2.

It is evident that the validity of claim 39 holds even if P_a uses *distinct* authentication keys for each of the parallel executions E_1 and E_2. In general, for any player P_i, running k parallel executions $E_1, E_2 \dots E_k$ of any (valid) ABG protocol, one can claim the following:

Claim 40 *If $\exists j$, $j \in (1 \dots k)$, such that, P_i is Byzantine corrupt in E_j, adversary can forge messages on behalf of P_i in E_l, $\forall l \in (1 \dots k)$, $l \neq j$, even though P_i is honest in E_l.*

Proof: Similar to the proof of Claim 39. ∎

Similar to claim 39, validity of claim 40 holds even if P_i uses *distinct* authentication keys for each of the parallel executions $E_1, E_2 \dots E_k$.

5.3.1 Determining the Fate of P_a in E_2

We return to the scenario considered prior to Claim 39. Since adversary can forge messages on behalf of P_a in E_2, a valid question at this point will be: *In execution E_2 should P_a be required to output a value same as decided upon by other honest players in E_2 ?* At a first glance the answer may be NO. The rationale being: In E_2, P_a has lost his private key(k_2) to the adversary, therefore, P_a is helping the adversary. We now present a series of arguments to contest as to why **should** P_a in E_2 output a value same as decided upon by other honest players in E_2 –

1. *Operating system perspective:* As discussed earlier, players running parallel executions can also be visualized as processors running multiple threads in parallel. It is well known that within an operating system, a thread(with administrative privileges) can always read the internal data of another thread. Thus, a faulty thread can always execute that code which will read the data internal to a non-faulty thread(a thread is said to be non-faulty if and only if it is not Byzantine corrupt). In such a scenario, a non-faulty thread cannot not be penalized as the data 'leakage' from the non-faulty thread to the adversary happens for no mistake of the non-faulty thread. Therefore, all non-faulty players(such as P_a in E_2 in our scenario) should output the same decision value in the corresponding execution.

2. *Passive Model:* Under parallel composition of executions, such as the example scenario considered here, by virtue of corrupting P_a in E_1 adversary can read the internal data of P_a in E_2. Thus, P_a can as well be treated as 'passively corrupt' in E_2. In general, if any player is Byzantine corrupt in any execution(s), then this player can be treated as passively corrupt in all other parallel executions. In section 4.3 we argued as to why should an ABG protocol ensure that passively corrupt players output a value same as decided upon by honest players. Therefore, the arguments presented in section 4.3 hold here as well.

3. *Composability of protocols:* Intuitively, the notion of composability of a (valid)protocol(for a given functionality) aims to achieve the designated functionality despite other executions in background. With respect to ABG, the functionality requires all non-faulty players to decide upon same value in every parallel execution. Here, a player is said to be non-faulty if and only he executes the designated protocol faithfully. Thus, P_a in E_2 output a value same as decided upon by other honest players in E_2.

5.4 Problem Definition

We directly adopt the definition of [LLR02, LLR06] to capture the notion of parallel composition of ABG protocols.

Definition 14 (Composable ABG [LLR02, LLR06]) *Let $p_1, \ldots, p_n$ be players for an ABG protocol Π. Then, Π remains secure under parallel composition if for every adversary $\mathcal{A}$, the requirements for ABG (which is elaborated in Definition 15) hold for Π for every execution within the following process: Repeat the following process in parallel until the adversary halts:*

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

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.* [LLR02, LLR06], Definition 14 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 define the requirements of ABG using the standard ideal/real process simulation paradigm.

Ideal process (Ψ_{ideal})

Participants: Ideal process consists of set $\mathbb{P}$ of n players including the General $\mathcal{G}$, incorruptible TTP (trusted third party) and an ideal process adversary $\mathcal{S}$.

Ideal process (Ψ_{ideal}) execution: We assume that all message transmissions in the following protocol are perfectly secure. The ideal process proceeds as follows:

1. $\mathcal{G}$ sends his value v to TTP and TTP forwards the same to $\mathcal{S}$.

2. TTP 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 $\mathcal{G}$ 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. $IDEAL_{TTP,\mathcal{S}}(v)$ denotes 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. $IDEAL_{TTP,\mathcal{S}}$ denotes the ensemble $\{IDEAL_{TTP,\mathcal{S}}(v)\}_{v\in\{0,1\}}$.

Real life process ($\Psi_{real}(\Pi)$)

*P*articipants: Real process consists of set $\mathbb{P}$ of n players including the General $\mathcal{G}$ and a real process adversary $\mathcal{A}$.

Real process (Ψ_{real}) execution: Unlike in the ideal process, here the players interact among themselves as per a designated protocol Π and the real process adversary $\mathcal{A}$. The real process proceeds as follows:

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 players under his control) 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 $\mathcal{G}$ 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 15 (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 real process adversary $\mathcal{A}$ that corrupts the players in I in $\Psi_{real}(\Pi)$, there exists an 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 similar.*

5.5 Corrupting Less Can Damage More!

While proving correctness of a given protocol, it common to assume that an adversary always uses his full power. This is because a protocol that can tolerate an adversary using full power can always tolerate an adversary using less of power. With respect to parallel executions also it appears that the a t-adversary is most powerful when it corrupts a set of t players in every parallel execution. Surprisingly, we show that this may not always be true i.e. it may not always be in the best interest of the adversary to corrupt players at full-throttle in every parallel execution.

We support our case by presenting 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 parallel execution. Consider an ABG protocol (with USIDs) over completely connected 3 players $\{a, b, c\}$ tolerating a 2-adversary that composes in parallel twice (existence of such a protocol is well known as $n > t$). W.l.o.g a is the General.

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 General a starts with

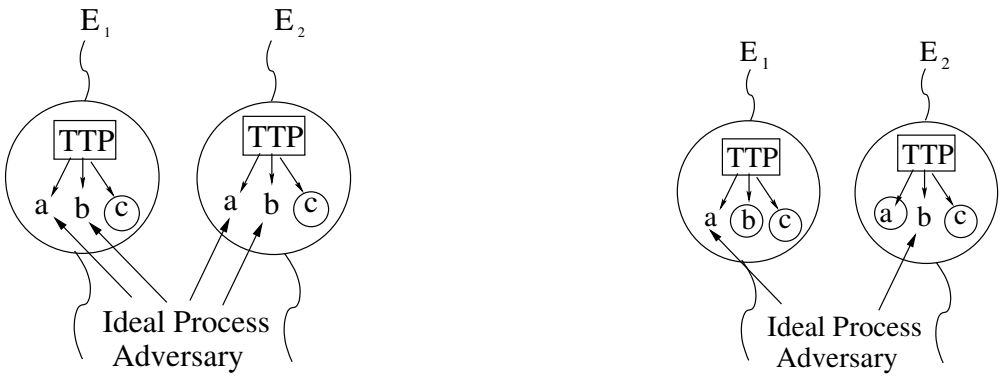

Figure 5.3: Corresponding ideal process execution for a scenarios s_1 and s_2.

input value 0. Consider the corresponding ideal process execution as shown in Figure 5.3. In ideal execution, player c (encircled) in both E_1 and E_2 is bound to receive correct value from the TTP. Then 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 General is corrupt).

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 with value 1 in E_2. Consider the corresponding ideal process execution as shown in Figure 5.3. In ideal execution, players b, c in E_1 and players a, c in E_2 receive correct value from the TTP. Then 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 General 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.

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 5.6, we prove that there does not exist any protocol(with USID) that composes in parallel twice and solves ABG for $n = 3$, $t = 2$. This implies that $n > t$ is *not* sufficient for parallel composition of ABG protocols(with USIDs). Rather in section 5.7 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 parallel executions, a t-adversary may corrupt a player in *some(or all)* the parallel executions. As mentioned previously, a player running multiple parallel executions can be visualized as a processor running parallel threads. Then, adversary can attack the player, execute a code different from the designated protocol in *some(or all)* of the threads. This is same as adversary corrupting this player in some(or all) executions. Therefore, a t-adversary may as well 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.

5.6 $n > t$ is not Sufficient for Parallel Composition of ABG

We now formally show that $n > t$ is not sufficient for parallel composition of ABG protocols(using USIDs). We substantiate our claim by proving that there *does not* exist any ABG protocol Π using USIDs that composes in parallel even twice over a completely connected network $\mathcal{N}$ (Figure 5.4) of 3 players $\mathbb{P}$=$\{A,B,C\}$ influenced by a 2-adversary (2-out-of-3). For the rest of this chapter we refer to a protocol Π using USIDs that composes in parallel k times and solves ABG[definition 14]

as $\Pi_{k,\ USID}$.

As a prelude we extend the definition of *view* (as defined in equation 4.2) to incorporate the fact that a given scenario may consists of multiple parallel executions. We define $msg_i^{E_l,\Omega}(a,b)_a$ denote the message sent by player a to player b in i^{th} round of execution E_l of scenario Ω. Then view of a player a at the end of round i in execution E_l of scenario Ω, denoted by $view_{a,i}^{E_l,\Omega}$, can be represented as

$$view_{a,i}^{E_l,\Omega} = \bigcup_k (msg_k^{E_l,\Omega}(x,a)_x),\ \forall k \in \{1 \dots i\},\ \forall x \in \mathbb{P} \tag{5.1}$$

On similar lines as Equation 4.3, we say that view of player a running in execution E_l of scenario Ω is same as view of player b running in execution E_m of scenario Γ iff:

$$view_{a,k}^{E_l,\Omega} \sim view_{b,k}^{E_m,\Gamma},\ \text{iff},\ msg_k^{E_l,\Omega}(x,a) \sim msg_k^{E_m,\Gamma}(x,b),\ \forall k \in (1 \dots i),\ \forall x \in \mathbb{P} \tag{5.2}$$

As shorthand we use E_l, Ω in order to refer to execution E_l of scenario Ω. We are now ready to state our theorem.

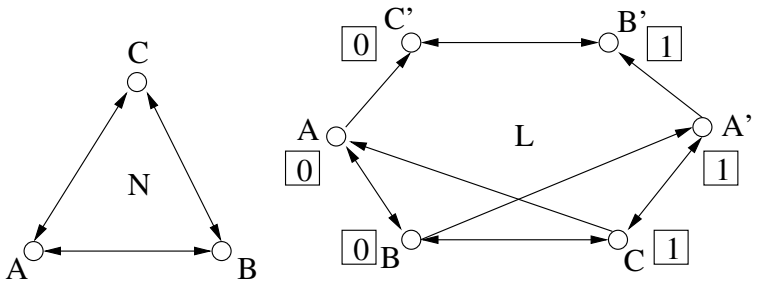

Figure 5.4: Network $\mathcal{N}$ and System L.

Theorem 41 *There does not exist any* $\Pi_{2,\ USID}$ *tolerating a* 2*-adversary over a completely connected network* $\mathcal{N}$ *of* 3 *nodes.*

Proof: We assume there exists a protocol $\Pi_{2,\ USID}$ over $\mathcal{N}$ 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 similar, thus violating Definition 14.

Using the proof technique developed by Fischer *et al.* [FLM85], we show that $\mathcal{A}$ can ensure that in one of the parallel executions of $\Pi_{2,\ USID}$, honest people exhibit contradictory behaviour. Using $\Pi_{2,\ USID}$ we create a protocol π'[Definition 16] in such a way that if $\Pi_{2,\ USID}$ exists then so does π'(Lemma 42). Using two copies of π' we construct a system L (as shown in Figure 5.4), and show that L must exhibit contradictory behaviour. This implies impossibility of the assumed protocol $\Pi_{2,\ USID}$.

We do not know what system L solves. Formally, L is a synchronous system with a well defined behaviour. That is, system L has a well defined output distribution for any particular input

assignment. We show that for a particular input assignment, no such well defined behaviour is possible. Further, no player in L knows the complete system. Each player in aware of only his immediate neighbours. In reality a player may be connected to either a or a', but it cannot differentiate between the two. It knows its neighbour only by its local name which may be a. Specifically, L is constructed in a such a way that the in-neighbourhood of any node a(or a') in L is same as in-neighbourhood of corresponding node a in $\mathcal{N}$.

Let E_1 and E_2 be two parallel executions of $\Pi_{2,\ USID}$ over $\mathcal{N}$. Let, α_1, α_2 and α_3 be three distinct scenarios as described below –

- α_1**:** In execution E_1, A is the General starting with input 0 and adversary $\mathcal{A}$ corrupts C in Byzantine fashion. In E_2, $\mathcal{A}$ corrupts A in Byzantine fashion.
- α_2**:** In E_1, A is the General. $\mathcal{A}$ corrupts A and interacts with B as if A started with input 0 and interacts with C as if A started with input 1.
- α_3**:** In execution E_1, A is the General starting with input 1 and $\mathcal{A}$ corrupts B in Byzantine fashion. In E_2, $\mathcal{A}$ corrupts A in Byzantine fashion.

Further, let α be an execution of L where each player starts with input value as shown in Figure 5.4. All the players in α are honest and follow the designated protocol correctly.

We claim that $\mathcal{A}$ can ensure that whatever *view* (as defined in equation 5.1) A(similarly B) gets in α, $\mathcal{A}$ can generate the same view for A(similarly B) in execution E_1 of scenario α_1 i.e. $view^{\alpha}_{A,i} \sim view^{E_1,\alpha_1}_{A,i}$(similarly $view^{\alpha}_{B,i} \sim view^{E_1,\alpha_1}_{B,i}$). This implies that the player A cannot ever differentiate between execution E_1 in scenario α_1 and α (dubbed $E_1, \alpha_1 \overset{A}{\sim} \alpha$). Similarly, player B cannot ever differentiate between execution E_1 in scenario α_1 and α ($E_1, \alpha_1 \overset{B}{\sim} \alpha$). From the definition of ABG [Definition 14], for E_1 in α_1, both A, B should decide on value 0. Since view of A(similarly B) is same in E_1, α_1 and α, both A, B in α will also decide on value 0. (We are able to make claims regarding the outputs of A and B in α as their views are same as those in E_1, α_1. Thus by analyzing their outputs in E_1, α_1, we can determine there outputs in α.) Similarly, we claim that $\mathcal{A}$ can ensure that whatever *view* A'(similarly C) gets in α, in E_3 $\mathcal{A}$ can generate the same view for A(similarly C) in α_3 i.e. $view^{\alpha}_{A',i} \sim view^{E_1,\alpha_3}_{A,i}$(similarly $view^{\alpha}_{C,i} \sim view^{E_1,\alpha_3}_{C,i}$). Thus, $E_1, \alpha_3 \overset{A'}{\sim} \alpha$ and $\alpha_3 \overset{C}{\sim} \alpha$. Both A, C in α_3 should decide on value 1. Then so will both A', C in E_1, α_3 in α. Similarly, we claim that $\mathcal{A}$ can ensure that $E_1, \alpha_2 \overset{B}{\sim} \alpha$ and $E_1, \alpha_2 \overset{C}{\sim} \alpha$. As per the definition of ABG, B, C in E_1, 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 behaviour.

To complete the proof we need to show that $\mathcal{A}$ can always ensure that – $A(B)$ gets same view in α and E_1, α_1, $B(C)$ gets same view in α and E_1, α_2 and $A(C)$ get same view in α and E_1, α_3. We prove the same in Lemma 44, 47, 48 respectively. As a prelude, we define the protocol π'[Definition 16] and show that if $\Pi_{2,\ USID}$ exists then so does π'(Lemma 42). ∎

Definition 16 (π') *For 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')* [2] *in π'. Similarly any statement of the kind "c sends message m to a" in $\Pi_{2,\ USID}$ is replaced by "c multicasts message m to all instances of a" in π'. Rest all statements in π' are same as those in $\Pi_{2,\ USID}$.*

Lemma 42 *If $\Pi_{2,\ USID}$ exists, then π' exists.*

Proof: Implied from Definition 16. ∎

To complete the proof of Theorem 41 we first show that $\mathcal{A}$ can always ensure that $A(B)$ get same view in α and E_1, α_1. We essentially show that for any round i, $\mathcal{A}$ can always ensure that $A(B)$ gets same messages in E_1, α_1 and α. From equation 5.2, it follows that $A(B)$ get same view in α and E_1, α_1. Intuitively, the validity of our claim can be seen from the following argument – since system L defined in Theorem 18 and Theorem 41 is same, then so is execution α. For the present case, note that in E_1, α_1 and E_2, α_1 adversary($\mathcal{A}'$) actively controls C and A respectively. By virtue of corrupting A actively in E_2, α_1, adversary can always read all the data of A in E_1, α_1. This is same as $\mathcal{A}'$ controlling A passively in E_1, α_1. In Lemma 20 we proved that an adversary($\mathcal{A}$) that corrupts C actively and controls A passively can ensure that $A(B)$ get same view in α and α_1. Then $\mathcal{A}'$ can always ensure that $A(B)$ get same view in α and E_1, α_1.
We now give the behaviour of adversary in E_1, α_1:

1. *Send outgoing messages of round i:* Based on the messages received during round $i-1$, $\mathcal{A}$ decides on the messages to be sent in round i. For round 1, $\mathcal{A}$ sends to B what an honest C would have sent to B in execution E_1, α_2. For $i \geq 2$, $\mathcal{A}$ authenticates $msg_{i-1}^{E_1,\alpha_1}(B,C)_B$ using C's key and sends it to A. For $msg_{i-1}^{E_1,\alpha_1}(A,C)_A$, $\mathcal{A}$ examines the message. If the message has not been authenticated by B even once, it implies that the message has not yet been seen by B. Then $\mathcal{A}$ authenticates and sends same message to B as C would have sent to B in round i of execution E_1, α_2. Formally, $\mathcal{A}$ constructs $msg_{i-1}^{E_1,\alpha_1}(A,C)_A$,($\mathcal{A}$ can construct $msg_{i-1}^{E_1,\alpha_1}(A,C)_A$, since A is corrupt in E_2, α_1 adversary can pull all the relevant data specifically - secret key, USID used in E_1, α_1, input value of A and messages received by A in previous rounds of E_1, α_1.) such that $msg_{i-1}^{E_1,\alpha_1}(A,C)_A \sim msg_{i-1}^{E_1,\alpha_2}(A,C)_A$, authenticates it using C's key and sends it to B. If the message has been authenticated by B even once, $\mathcal{A}$ simply authenticates $msg_{i-1}^{E_1,\alpha_1}(A,C)_A$ using C's key and sends it to B.

2. *Receive incoming messages of round i:* $\mathcal{A}$ obtains messages $msg_i^{E_1,\alpha_1}(A,C)_A$ and $msg_i^{E_1,\alpha_1}(B,C)_B$ via C. (These are round i messages sent by A and B respectively to C). Similarly via A in E_2, α_1, $\mathcal{A}$ obtains messages $msg_i^{E_1,\alpha_1}(B,A)_B$ and $msg_i^{E_1,\alpha_1}(C,A)_C$. (These are also round i messages sent by B and C respectively to A in E_1. Players respectively compute these messages according to their input, secret key, protocol run by them and the view they get up to round $i-1$).

Consider execution α from the perspective of A(similarly B). We now show that messages received by $A(B)$ in round i of α are same as messages received by $A(B)$ in round i of E_1, α_1 respectively.

Lemma 43 *$msg_i^{\alpha}(x,A)_x \sim msg_i^{E_1,\alpha_1}(x,A)_x$ and $msg_i^{\alpha}(x,B)_x \sim msg_i^{E_1,\alpha_1}(x,B)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.*

Proof: The proof stems from the fact that since A is corrupt in E_2, α_1 adversary can pull(refer to Claim 40) private data of E_1, α_1. This is same treating A as passively corrupt in E_1, α_1. Therefore rest of the proof is on similar lines as proof of Lemma 20. Details omitted. ∎

[2] a and a' are independent copies of a with same authentication key.

Lemma 44 *$view_A^{\alpha} \sim view_A^{E_1,\alpha_1}$ and $view_B^{\alpha} \sim view_B^{E_1,\alpha_1}$*

Proof: Follows from equation 5.2 and Lemma 43. ∎

Similarly, proofs of Lemma 45, 46 follow from the proofs of Lemma 22, 24 respectively. We only state our Lemmas.

Lemma 45 *$msg_i^{\alpha}(x,B)_x \sim msg_i^{E_1,\alpha_2}(x,B)_x$ and $msg_i^{\alpha}(x,C)_x \sim msg_i^{E_1,\alpha_2}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$*

Lemma 46 *$msg_i^{\alpha}(x,C)_x \sim msg_i^{E_1,\alpha_3}(x,C)_x$ and $msg_i^{\alpha}(x,A')_x \sim msg_i^{E_1,\alpha_3}(x,A)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$*

Lemma 47 *$view_B^{\alpha} \sim view_B^{E_1,\alpha_2}$ and $view_C^{\alpha} \sim view_C^{E_1,\alpha_2}$*

Proof: Follows from equation 5.2 and Lemma 45. ∎

Lemma 48 *$view_C^{\alpha} \sim view_C^{E_1,\alpha_3}$ and $view_{A'}^{\alpha} \sim view_A^{E_1,\alpha_3}$.*

Proof: Follows from equation 5.2 and Lemma 46. ∎

5.7 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 network $\mathcal{N}'$ (Figure 5.5) of four nodes $\mathbb{P} = \{A, B, C, D\}$ tolerating adversary basis $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. 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 parallel execution. The proof technique is similar to one used for proving impossibility of 2-out-of-3 (Theorem 41) in section 5.6.

Theorem 49 *There does not exist any protocol $\Pi_{2,\,USID}$ over a complete network $\mathcal{N}'$ of four nodes $\mathbb{P} = \{A, B, C, D\}$, tolerating adversary basis $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$.*

Proof: We assume there exists a protocol $\Pi_{2,\,USID}$ tolerating adversary basis $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$ over a complete network $\mathcal{N}'$ (Figure 5.5). We show that there exist two parallel executions of $\Pi_{2,\,USID}$, where the real process adversary $\mathcal{A}$ (characterized by $\mathbb{A}$) 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 similar, thus violating Definition 14.

Similar to proof of Theorem 41, 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 L' (as shown in Figure 5.5), and show that L' must exhibit contradictory behaviour. This contradicts our assumption about existence of $\Pi_{2,\,USID}$.

We do not know what system L' solves. All we know is that L' is a synchronous system with a well defined behaviour. That is, L' has a well defined output distribution for any particular input assignment. We show that for a particular input assignment, no such well defined behaviour is possible. Further no player in L' knows the complete system. Each player in aware of only his immediate neighbours. In reality a player may be connected to either a or a', but it cannot differentiate between the two. It knows its neighbour only by its local name which may be a.

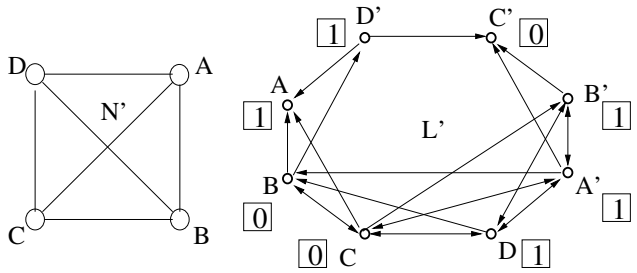

Figure 5.5: Network $\mathcal{N}'$ and System L'.

Further, in-neighbourhood of any node a(or a') in L' is same as in-neighbourhood of corresponding node a in $\mathcal{N}'$.

Let E_1, E_2 be two parallel executions of $\Pi_{2,\ USID}$ over $\mathcal{N}'$. Let β_1, β_2 and β_3 be three distinct scenarios as described below –

- β_1**:** In execution E_1, B is the General starting with input 0 and adversary $\mathcal{A}$ corrupts A,D in Byzantine fashion. In execution E_2, adversary corrupts B in Byzantine fashion.
- β_2**:** In execution E_1, B is the General. Adversary corrupts B and makes interacts with C as if B started with input value 0 & interacts with A, D as if B started with input value 1. In execution E_2, adversary corrupts A in Byzantine fashion.
- β_3**:** In execution E_1, D is the General starting with input 1 and adversary corrupts C in Byzantine fashion. In execution E_2, adversary corrupts B in Byzantine fashion.

Further, let β be an execution of L' where each player starts with input value as shown in Figure 5.5. All the players in β are honest and follow the designated protocol correctly.

We claim that $\mathcal{A}$ can ensure that whatever *view* (as defined in equation 5.1) B(similarly C) gets in β, $\mathcal{A}$ can generate the same view for B(similarly C) in execution E_1 of scenario β_1 i.e. $view^{\beta}_{B,i} \sim view^{E_1,\beta_1}_{B,i}$(similarly $view^{\beta}_{C,i} \sim view^{E_1,\beta_1}_{C,i}$). This implies that the player B cannot ever differentiate between execution E_1 in scenario β_1 and β (dubbed $E_1, \beta_1 \overset{B}{\sim} \beta$). Similarly, player C cannot ever differentiate between execution E_1 in scenario β_1 and β ($E_1, \beta_1 \overset{C}{\sim} \beta$). From the definition of ABG [Definition 14], in E_1, β_1, both B, C should decide on value 0. Since view of $B(C)$ is same in E_1, β_1 and β, both B, C in β will also decide on value 0. (We are able to make claims regarding the output of $B(C)$ in β as its view is same as that in E_1, β_1. Thus by analyzing its output in E_1, β_1, we can determine its output in β.) Similarly, we claim that $\mathcal{A}$ can ensure that $view^{\beta}_{A',i} \sim view^{E_1,\beta_3}_{A,i}$, $view^{\beta}_{B',i} \sim view^{E_1,\beta_3}_{B,i}$ and $view^{\beta}_{D,i} \sim view^{E_1,\beta_3}_{D,i}$. Thus, $E_1, \beta_3 \overset{A'}{\sim} \beta$, $E_1, \beta_3 \overset{B'}{\sim} \beta$ and $E_1, \beta_3 \overset{D}{\sim} \beta$. A, B and D in E_1, β_3 should decide on value 1. Then so will A', B' and D in β. Similarly, we claim that $\mathcal{A}$ can ensure that $E_1, \beta_2 \overset{C}{\sim} \beta$, $E_1, \beta_2 \overset{D}{\sim} \alpha$ and $E_1, \beta_2 \overset{A'}{\sim} \beta$. As per the definition of ABG[Definition 14], C, D and A in E_1, β_2 should agree on same value, then so should C, D and A' in β. But in β, C and D, A' have already decided on 0 and 1 respectively. This implies L' must exhibit contradictory behaviour.

To complete the proof all we need to show is that $\mathcal{A}$ can always ensure the following – $B(C)$ gets same view in β and E_1, β_1, $C(D,\ A')$ gets same view in β and E_1, β_2 and $A'(B',D)$ gets same view in β and E_1, β_3. We proof the same in Lemma 52, 55, 56 respectively. As a prelude, we define the protocol η' [Definition 17] and show that if $\Pi_{2,\ USID}$ exists then so does η'(Lemma 50). ∎

Definition 17 (η') *All statements in η' are same as those in $\Pi_{2,\ USID}$ except the following:*

- *any statement in $\Pi_{2,\ USID}$ of the kind "A sends message m to B" is replaced by "A multicasts message m to all instances of B"(i.e. B,B')* [3]*. Similarly, "A sends message m to C" is replaced by "A multicasts message m to all instances of C"(i.e. C,C').*
- *any statement in $\Pi_{2,\ USID}$ of the kind "B sends message m to D" is replaced by "B multicasts message m to all instances of D"(i.e. D,D').*
- *any statement in $\Pi_{2,\ USID}$ of the kind "C sends message m to A" is replaced by "C multicasts message m to all instances of A"(i.e. A,A'). Similarly, "C sends message m to B" is replaced by "C multicasts message m to all instances of B"(i.e. B,B').*
- *any statement in $\Pi_{2,\ USID}$ of the kind "D sends message m to B" is replaced by "D multicasts message m to all instances of B"(i.e. B,B').*

Lemma 50 *If $\Pi_{2,\ USID}$ exists then η' exists.*

Proof: Implied from Definition 17. ∎

Lemma 51 $msg_i^{\beta}(x,B)_x \sim msg_i^{E_1,\beta_1}(x,B)_x$ *and* $msg_i^{\beta}(x,C)_x \sim msg_i^{E_1,\beta_1}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.

Proof: The proof stems from the fact that since B is corrupt in E_2, β_1, thus adversary can pull(refer to Claim 40) private data of E_1, β_1. This is same treating B as passively corrupt in E_1, β_1. Therefore rest of the proof is on similar lines as proof of Lemma 28. Details omitted. ∎

Lemma 52 $view_B^{\beta} \sim view_B^{E_1,\beta_1}$ *and* $view_C^{\beta} \sim view_C^{E_1,\beta_1}$

Proof: Follows from equation 5.2 and Lemma 51. ∎

Similarly, proofs of Lemma 53, 54 follow from the proofs of Lemma 30, 32 respectively. We only state our Lemmas.

Lemma 53 $msg_i^{\beta}(x,C)_x \sim msg_i^{E_1,\beta_2}(x,C)_x$, $msg_i^{\beta}(x,D)_x \sim msg_i^{E_1,\beta_2}(x,D)_x$ *and* $msg_i^{\beta}(x,A')_x \sim msg_i^{E_1,\beta_2}(x,A)_x$ $\forall i > 0$, $\forall x \in \mathbb{P}$.

Lemma 54 $msg_i^{\beta}(x,A')_x \sim msg_i^{E_1,\beta_3}(x,A)_x$, $msg_i^{\beta}(x,B')_x \sim msg_i^{E_1,\beta_3}(x,B)_x$, *and* $msg_i^{\beta}(x,D)_x \sim msg_i^{E_1,\beta_3}(x,D)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.

Lemma 55 $view_C^{\beta} \sim view_C^{E_1,\beta_2}$, $view_D^{\beta} \sim view_D^{E_1,\beta_2}$, $view_{A'}^{\beta} \sim view_A^{E_1,\beta_2}$

Proof: Follows from equation 5.2 and Lemma 53. ∎

Lemma 56 $view_{A'}^{\beta} \sim view_A^{E_1,\beta_3}$, $view_{B'}^{\beta} \sim view_B^{E_1,\beta_3}$, $view_D^{\beta} \sim view_D^{E_1,\beta_3}$.

Proof: Follows from equation 5.2 and Lemma 54. ∎

We now present the main theorem of this chapter.

[3] B and B' are independent copies of B with same authentication key.

Theorem 57 (Main Theorem) *There exists a protocol $\Pi_{k,\ USID}$ tolerating t-adversary over a completely connected network $\mathcal{N}$ 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 network 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 can corrupt 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 $n < 2t$.

We assume that there exists a protocol $\eta_{2,USID}$ over a complete network $\mathcal{N}$ of n nodes tolerating a (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 network of four nodes $\{A, B, C, D\}$, tolerating adversary basis $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. We then show that if $\eta_{2,USID}$ satisfies definition 14, then so does $\Pi_{2,\ USID}$. But this contradicts Theorem 49. Thus our assumption that there exists a solution $\eta_{2,USID}$ for $n \leq 2t_1 + min(t_1, t_2)$, $t_2 > 0$ 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 14 when $n \leq 2t_1 + min(t_1, t_2)$, $t_2 > 0$, then so does $\Pi_{2,\ USID}$ tolerating $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. Consider two honest players i and j ($i \neq j$) in execution E_i'. Each of them simulates at least 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 $\mathcal{G}$ 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 14 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 14 and tolerates $\mathbb{A} = \{((A, D), (B)), ((B), (A)), ((C), (B))\}$. But from Theorem 49, 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 claim that for $n \geq 2t$, the *EIGPrune* protocol presented in section 4.6.1 when augmented with USIDs is a valid ABG protocol that composes for any number of parallel executions.

We substantiate our claim in the following manner – From Lemma 38, it is evident that the protocol works correctly for $n > 2t_b + t_p$. Using $t_b = t - 1$ and $t_p = 1$ in Theorem 34, one gets that *EIGPrune* protocol works correctly for $n \geq 2t$. To prove that the protocols remains secure under parallel composition, we use the proof technique developed by Lindell *et al.* [LLR02, LLR06] wherein the security of the protocol under composition is reduced to security of the protocol in stand alone setting.

Let $\pi(1)\ldots\pi(l)$ be l parallel executions of the *EIGPrune* protocol. We now prove that if there exists an adversary $\mathcal{A}$ 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 *EIGPrune* protocol. This violates Lemma 38. But we know that Lemma 38 is true. This contradicts our assumption that there exists an adversary $\mathcal{A}$ that can attack and succeed in some execution $\pi(i)$, $i \in (1,l)$.

Let execution $\pi(id_i)$ use unique session identifier id_i. Let there exists an adversary $\mathcal{A}$ that succeeds in some execution $\pi(i)$. $i \in (1,l)$. For authentication, players use a secure signature scheme $((Gen,S_{id},V_{id}),S_{\neg id})$ [Refer to section 2.1.5].

Using $\mathcal{A}$ we construct adversary $\mathcal{A}'$ that is bound to succeed against stand alone execution $\Pi(id)$ of *EIGPrune* protocol. 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 parallel 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 $\mathcal{A}'$ has 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 parallel executions for $\mathcal{A}$ is perfect. Thus if $\mathcal{A}$ succeeds in breaking $\pi(id_i)$, then $\mathcal{A}'$ should also succeed in breaking $\Pi(id)$. This clearly contradicts Lemma 38. ∎

5.8 On Contradiction with Literature

We now elaborate on the shortcoming in the proof for sufficiency of $n > t$ for protocols with USIDs for parallel composition of ABG [LLR02, LLR06]. A brief overview of the proof for sufficiency of $n > t$ for protocols with USIDs for parallel composition of ABG in presented in section 5.8.1. Reader is encouraged to read section 5.8.1 before proceeding further.

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.

The proof assumes that under parallel 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 the particular player (who is honest in $\pi(id_k)$) in some

other execution, $\mathcal{A}$ can gain access to his private in execution $\pi(id_k)$ and thus forge messages on behalf of him 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 $\pi(id_l)$. By virtue of corrupting p_1 in $\pi(id_l)$, $\mathcal{A}$ can gain access to his private key in $\pi(id_k)$[follow from Claim 40]. With this key $\mathcal{A}$ can always forge messages on behalf of p_1 in $\pi(id_k)$.

We proceed to show that the simulation by an adversary $\mathcal{A}'$ attacking a stand alone execution of $\pi(id)$ of an adversary $\mathcal{A}$ attacking parallel executions $\pi(id_1) \dots \pi(id_l)$ is *not perfect* as claimed in the proof. Let $\mathcal{A}$ attack $\pi(id_1) \dots \pi(id_l)$ and succeeds in some execution say $\pi(id_i)$. W.l.o.g let P_a be an honest player in execution $\pi(id_i)$. Note that by corrupting P_a in some other parallel execution, say $\pi(id_k)$ $k \in (1 \dots l)$ $k \neq i$, $\mathcal{A}$ can forge messages on behalf of P_a in $\pi(id_i)$(From claim 40). Now $\mathcal{A}'$ simulates $\mathcal{A}$ by internally incorporating all the parallel executions $\pi(id_1) \dots \pi(id_l)$ except $\pi(id_i)$. In $\pi(id_i)$, $\mathcal{A}'$ externally interacts with the honest players and passes messages between them and $\mathcal{A}$.

Before proceeding further we observe the following – claim 40 stems from the fact that parallel threads are running within the same physical memory, so a faulty thread can always pull information private to a non-faulty thread. We now claim that in stand alone execution $\pi(id)$, $\mathcal{A}'$ cannot forge messages on behalf P_a. The claim stems from the following observation – even though $\mathcal{A}'$ corrupts P_a in execution $\pi(id_k)$ ($\mathcal{A}'$ is internally simulating $\pi(id_k)$), $\mathcal{A}'$ *cannot* gain access to private key used by P_a in execution $\pi(id)$. This is because threads $\pi(id)$ and $\pi(id_k)$ are running in physically different memory. Thus, $\mathcal{A}'$ cannot forge messages on behalf of P_a in $\pi(id)$ unlike $\mathcal{A}$. Thus, the simulation is *not perfect* as claimed otherwise.

Remark: Claim 40 is central to various proofs and arguments presented in this chapter. As pointed out earlier too, Claim 40 holds true even if every player uses a different set of authentication and verification keys for each of the parallel execution. Therefore, the results presented in this chapter hold true if every player uses a different set of authentication and verification keys for each of the parallel execution of a (valid)ABG protocol.

5.8.1 Overview of the proof given in Literature

The proof for sufficiency of $n > t$ for protocols with USIDs for parallel composition of ABG [LLR02, LLR06] 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.

The proof defines 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, $\mathrm{V}(vk,m,\mathrm{S}(sk,m)) = 1$, 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 $\mathrm{V}(vk,m^*,\sigma^*) = 1$ 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 $\mathrm{Aux}(sk,.)$. 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 $\mathrm{Aux}(sk,\cdot) = S_{\neg id}(sk,m)$

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, it assumes π to be a secure protocol for ABG using signature scheme (Gen,S,V). It defines modified protocol $\pi(id)$ to be exactly same as π except that it uses signature scheme (Gen,S_{id},V_{id}) as defined above. The proof then proceeds to show that the scheme $((Gen,S_{id},V_{id}), S_{\neg id})$ 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, the proof demonstrates as to how an adversary $\mathcal{A}'$ for a single execution of $\pi(id)$ can simulate an adversary $\mathcal{A}$ for parallel executions $\pi(id_1) \ldots \pi(id_l)$, thus reducing the security of parallel executions to security of stand alone execution. If $\mathcal{A}$ attacks some parallel execution 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 $\pi(id_i)$. This contradicts the well known result of $n > t$ for ABG in stand alone model. [PSL80].

Chapter 6

Conclusion and Future Work

As highlighted in section 2.1.5, all known authentication schemes are based on hardness assumptions. Further, proofs of the hardness of these problems appear to be beyond the reach of contemporary mathematics. Thus, some of the schemes may subsequently turn out to be insecure. Then, apart from forging signatures of malicious players(t_b), the adversary can forge signatures of players(t_p) who's signature scheme are no more secure. An elegant method to (partially) deal with such a scenario is the concept of *robust combiners* [MPW07, HKN$^+$05, MP06]. Analogously, in the context of ABG, it is desirable to construct protocols that are guaranteed to be correct as long as the signature schemes of no more than any k players are insecure (in that execution of the protocol). Throughout the thesis, we work with deterministic protocols by which we mean that the protocol must have a zero-error probability in any run, passive corruption also models the above mentioned scenario where signature schemes of up to any t_p players may malfunction during the execution of the ABG protocol. Thus, our results of ABG under the influence of mixed adversary can also be seen in the light robust combiners for signatures schemes used in ABG protocols.

Further, the folklore has been that use of authentication reduces the problem of simulating a broadcast in presence to Byzantine faults to fail-stop failures. Thus, the protocols designed for fail-stop faults can be quickly adapted to solve ABG. However we have shown that this does not hold true for the case of ABG_{mix}. In a way, the problem of ABG_{mix} covers the entire spectrum of problems with ABG and BGP as two extremes. Consequentially, the protocols for ABG_{mix} borrow ideas from both ABG and BGP. From our results of $n > 2t_b + \min(t_b, t_p)$, it appears that studying this problem for *non-threshold* adversary will be interesting. Further, the problem is expected to yield interesting results over other networks such as *undirected incomplete networks, directed networks, hypergraphs, directed hypergraphs* in presence of threshold/non-threshold adversary.

With respect to parallel composition of ABG protocols, unique session identifiers aid in improving the fault-tolerance 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 [Can01a]. The anomaly with respect to ABG, as pointed out in Section 5.5, 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.

Bibliography

[AFM99] 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.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*, pages 137–156, 2007.

[BDP97] Piotr Berman, Krzysztof Diks, and Andrzej Pelc. Reliable broadcasting in logarithmic time with byzantine link failures. *J. Algorithms*, 22(2):199–211, 1997.

[BGP89] P. Berman, J. A. Garay, and K. J. Perry. Towards Optimal Distributed Consensus. In *Proceedings of the 21st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 410–415, 1989.

[BGP92a] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Bit optimal distributed consensus. pages 313–321, 1992.

[BGP92b] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal early stopping in distributed consensus (extended abstract). In *WDAG '92: Proceedings of the 6th International Workshop on Distributed Algorithms*, pages 221–237, London, UK, 1992. Springer-Verlag.

[BL87] James E. Burns and Nancy A. Lynch. The Byzantine Firing Squad Problem. In *Advances in Computing Research, Parallel and Distributed Computing*, volume 4, pages 147–161. JAI Press, Inc., 1987.

[BNDDS87] 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.

[BO83] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC '83: Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, New York, NY, USA, 1983. ACM.

[BO90] M. Ben-Or. Randomized Agreement Protocols. pages 72–83, 1990.

[Bor95] 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.

[Bor96a] 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.

[Bor96b] 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.

[BPC+08] Ashwinkumar B.V, Arpita Patra, Ashish Choudhary, Kannan Srinathan, and Chandrasekharan Pandu Rangan. On tradeoff between network connectivity, phase complexity and communication complexity of reliable communication tolerating mixed adversary. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 115–124, New York, NY, USA, 2008. ACM.

[Bra85] Gabriel Bracha. An o(lg n) expected rounds randomized byzantine generals protocol. In *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*, pages 316–326, New York, NY, USA, 1985. ACM.

[Bra87] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[Can96] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, The Weizmann Institute of Science, 1996.

[Can01a] 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`.

[Can01b] Ran Canetti. A unified framework for analyzing security of protocols. *Electronic Colloquium on Computational Complexity (ECCC)*, 8(16), 2001.

[CCD88] D. Chaum, C. Crepeau, and I. Damgard. Multi-party Unconditionally Secure Protocols. In *Proceedings of 20th Symposium on Theory of Computing (STOC)*, pages 11–19. ACM Press, 1988.

[CDDS89] B A Coan, D Dolev, C Dwork, and L Stockmeyer. The distributed firing squad problem. In *SIAM Journal on Computing*, volume 18(5), pages 990–1012, 1989.

[CF01] 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.

[CFF+05] Jeffrey Considine, Matthias Fitzi, Matthew K. Franklin, Leonid A. Levin, Ueli M. Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *J. Cryptology*, 18(3):191–217, 2005.

[CIO98] Giovanni Di Crescenzo, Yuval Ishai, and Rafail Ostrovsky. Non-interactive and non-malleable commitment. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 141–150, New York, NY, USA, 1998. ACM.

[CK02] 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.

[CKPR01] 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.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 494–503, New York, NY, USA, 2002. ACM.

[CMS89] Benny Chor, Michael Merritt, and David B. Shmoys. Simple constant-time consensus protocols in realistic failure models. *J. ACM*, 36(3):591–614, 1989.

[Coa87] B. A. Coan. *Achieving consensus in fault-tolerant distributed computer systems: protocols, lower bounds, and simulations.* PhD thesis, Cambridge, MA, USA, 1987.

[CPA+08] Ashish Choudhary, Arpita Patra, B. V. Ashwinkumar, K. Srinathan, and C. Pandu Rangan. Perfectly Reliable and Secure Communication Tolerating Static and Mobile Mixed Adversary. In Reihaneh Safavi-Naini, editor, *ICITS*, volume 5155 of *Lecture Notes in Computer Science*, pages 137–155. Springer, 2008.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, New York, NY, USA, 1993. ACM.

[CR03] 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.

[CW92] Brian A. Coan and Jennifer L. Welch. Modular construction of a Byzantine agreement protocol with optimal message bit complexity. *Inf. Comput.*, 97(1):61–85, 1992.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 542–552, New York, NY, USA, 1991. ACM.

[DFF+82] Danny Dolev, Michael J. Fischer, Rob Fowler, Nancy A. Lynch, and Raymond H. Strong. An Efficient Algorithm for Byzantine Agreement without Authentication. *Information and Control*, 52(3):257–274, 1982.

[DLM82] Richard A. DeMillo, Nancy A. Lynch, and Michael J. Merritt. Cryptographic protocols. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 383–400, New York, NY, USA, 1982. ACM.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment: crash failures. *Inf. Comput.*, 88(2):156–186, 1990.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[DRS90] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.

[DS82] Danny Dolev and H. Raymond Strong. Polynomial algorithms for multiple processor agreement. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 401–407, New York, NY, USA, 1982. ACM Press.

[DS83] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[Fel89] P. Feldman. Asynchronous Byzantine agreement in constant expected time. *Manuscript*, 1989.

[FF00] Marc Fischlin and Roger Fischlin. Efficient non-malleable Commitment Schemes. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 413–431, London, UK, 2000. Springer-Verlag.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220, New York, NY, USA, 2003. ACM.

[FLM85] 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.

[FLP85] 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.

[FM85] P. Feldman and S. Micali. Byzantine agreement in constant expected time (and trusting no one). In *FOCS' 85: Proceedings of the twenty-sixth Annual IEEE Symposium on the Foundations of Computer Science*, 1985.

[FM97] Pesech Feldman and Silvio Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[FM00a] Matthias Fitzi and Ueli Maurer. Global broadcast by broadcasts among subsets of players. In *IEEE International Symposium on Information Theory — ISIT 2000*, page 267. IEEE, June 2000.

[FM00b] 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.

[Gam85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Gen96] R. Gennaro. *Theory and Practice of Verifiable Secret Sharing*. PhD thesis, Massachusetts Institute of Technology (MIT), Cambridge, May 1996.

[GK96] Oded Goldreich and Hugo Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Comput.*, 25(1):169–192, 1996.

[GLR95] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults, 1995.

[GM93] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement in t + 1 rounds. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 31–41, New York, NY, USA, 1993. ACM Press.

[GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for n ¿ 3t processors in t + 1 rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.

[GM00] J. A. Garay and P. MacKenzie. Concurrent oblivious transfer. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 314, Washington, DC, USA, 2000. IEEE Computer Society.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game. In *Proceedings of the 19th Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.

[Gol04a] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[Gol04b] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2004.

[GP90] O. Goldreich and E. Petrank. The best of both worlds: guaranteeing termination in fast randomized byzantine agreement protocols. *Inf. Process. Lett.*, 36(1):45–49, 1990.

[GP92] 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*, volume 647 of *Lecture Notes in Computer Science (LNCS)*, pages 153–165, London, UK, 1992. Springer-Verlag.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 197–206, New York, NY, USA, 2008. ACM.

[Had83] V. Hadzilacos. Byzantine agreement under restricted types of failures (not telling the truth is different from telling lies). Technical Report Technical Report TR.CRCT TR-1, Harvard University, 1983.

[HH91] Vassos Hadzilacos and Joseph Y. Halpern. Message-optimal protocols for byzantine agreement (extended abstract). In *PODC '91: Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 309–323, New York, NY, USA, 1991. ACM.

[HH93] Vassos Hadzilacos and Joseph Y. Halpern. The failure discovery problem. *Mathematical Systems Theory*, 26(1):103–129, 1993.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In *EUROCRYPT*, pages 96–113, 2005.

[HM00] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13:31–60, 2000.

[KA94] R.M. Kieckhafer and M.H. Azadmanesh. Reaching Approximate Agreement with Mixed Mode Faults. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):53–63, 1994.

[KK07] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. 2007.

[KY84] A. Karlin and A.C. Yao. Manuscript. 1984.

[Lam83] L. Lamport. The weak byzantine generals problem. *J. ACM*, 30(3):668–676, 1983.

[LF82] Leslie Lamport and Michael J. Fischer. Byzantine generals and transactions commit protocols. Technical Report Opus 62, Menlo Park, California, 1982.

[Lin02] Yehuda Lindell. *On the Composition of Secure Multi-Party Computation*. PhD thesis, The Weizmann Institute of Science, May 2002.

[Lin03a] Y. Lindell. *Composition of Secure Multi-Party Protocols: A Comprehensive Study*, volume 2815 of *Lecture Notes in Computer Science (LNCS)*. Springer–Verlag, 2003.

[Lin03b] Yehuda Lindell. Bounded-concurrent Secure Two-party Computation without setup assumptions. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 683–692, New York, NY, USA, 2003. ACM.

[Lin03c] Yehuda Lindell. General Composition and Universal Composability in Secure Multi-Party Computation. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 394, Washington, DC, USA, 2003. IEEE Computer Society.

[LLR02] 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.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *J. ACM*, 53(6):881–917, 2006.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Lyn96] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Mateo, CA, USA, 1996.

[MN82] M.J.Fischer and N.A.Lynch. A Lower Bound on the Time to Assure Interactive Consistency. *Information Processing Letters*, 14:183–186, 1982.

[MP91] F. J. Meyer and D. K. Pradhan. Consensus with dual failure modes. *IEEE Trans. Parallel Distrib. Syst.*, 2(2):214–222, 1991.

[MP06] Remo Meier and Bartosz Przydatek. On robust combiners for private information retrieval and other primitives. In *CRYPTO*, pages 555–569, 2006.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, Parts i and ii. *Inf. Comput.*, 100(1):1–40, 1992.

[MPW07] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In *TCC*, pages 404–418, 2007.

[MSA88] M.Ben-Or, S.Goldwasser, and A.Wigderson. Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation. In *Proceedings of the 20th Symposium on Theory of Computing (STOC)*, pages 1–10. ACM Press, 1988.

[MT07] Achour Mostefaoui and Gilles Trédan. Towards the minimal synchrony for byzantine consensus. In *PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 314–315, New York, NY, USA, 2007. ACM.

[Nei94] Gil Neiger. Distributed consensus revisited. *Inf. Process. Lett.*, 49(4):195–201, 1994.

[NY90] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 427–437, New York, NY, USA, 1990. ACM.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *PODC '91: Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, New York, NY, USA, 1991. ACM.

[PCSR07] Arpita Patra, Ashish Choudhary, Kannan Srinathan, and C. Pandu Rangan. Perfectly Reliable and Secure Communication in Directed Networks Tolerating Mixed Adversary. In *DISC*, pages 496–498, 2007.

[PP05] Andrzej Pelc and David Peleg. Feasibility and complexity of broadcasting with random transmission failures. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 334–341, New York, NY, USA, 2005. ACM.

[PR03] Rafael Pass and Alon Rosen. Bounded-concurrent Secure Two-party Computation in a Constant Number of Rounds. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 404, Washington, DC, USA, 2003. IEEE Computer Society.

[PS04] 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.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, 1980.

[Rab83a] M. O. Rabin. Randomized Byzantine Generals. In *Proc. of the 24th Annu. IEEE Symp. on Foundations of Computer Science*, pages 403–409, 1983.

[Rab83b] Michael O. Rabin. Randomized byzantine generals. In *SFCS '83: Proceedings of the 24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409, Washington, DC, USA, 1983. IEEE Computer Society.

[RB89] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In *Proceedings of the 21st Symposium on Theory of Computing (STOC)*, pages 73–85. ACM Press, 1989.

[Reg04] Oded Regev. New lattice-based cryptographic constructions. *J. ACM*, 51(6):899–942, 2004.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 433–444, London, UK, 1992. Springer-Verlag.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21:120–126, February 1978.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Sha94] Adi Shamir. Efficient signature schemes based on birational permutations. In *CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 1–12, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[SKR02] K. Srinathan, M.V.N.A. Kumar, and C. Pandu Rangan. Asynchronous Secure Communication Tolerating Mixed Adversaries. In *Proceedings of ASIACRYPT '02*, volume 2501 of *Lecture Notes in Computer Science*, pages 224–242. Springer-Verlag, 2002.

[ST87] T. K. Srikanth and S. Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2(2):80–94, 1987.

[SW04] 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.

[TC84] R. Turpin and B. A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, Feb. 1984.

[Tou84] Sam Toueg. Randomized byzantine agreements. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 163–178, New York, NY, USA, 1984. ACM.

[TPS87] Sam Toueg, Kenneth J. Perry, and T. K. Srikanth. Fast distributed agreement. *SIAM J. Comput.*, 16(3):445–457, 1987.

[Yao82] Andrew Chi-Chih Yao. Protocols for Secure Computations. In *Proceedings of 23rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 160–164. IEEE Press, 1982.