Agreement can be Easier than Point-to-Point Communication

Prasant Gopal
prasant@research.iiit.ac.in

International Institute of Information Technology
Hyderabad, India
May 2009

FIRST DRAFT

To Isha

"*All my life I had the choice between hate and love. I chose love and here I am.*"

- Rahman on winning the Oscars

Acknowledgments

First of all, I would like to thank Dr. Kannan Srinathan who has been having a tremendous influence on my professional development. He introduced me to Distributed Computing, theoretical computer science and has been closely guiding my first steps. On top of being a devoted advisor, and a dear friend, he is an excellent teacher far beyond the ordinary. My interactions with him have greatly influenced my understanding of research and of life in general. His feedback and encouragement have greatly helped me to keep my spirits up.

More than anything else, if it was not for Dr. Srinathan and his patience at making me understand the things – I would not have gone this far. I can proudly say that Srinathan, his philosophy(A philosophy on which one can write an entire book on) and his interactions with me are what I cherish the most. It was Srinathan who taught me what should one pick from the environment and more importantly, what should not be picked! Rather, it will be appropiate to say - Srinathan is my *academic parent*. He has sown in me the invaluable seeds of inculating freedom of thought. It might be that destiny had it or it was a mere coincidence in that - Whenever there were clouds of doubts hanging over my head, Srinathan would appear from his hermit cave and blow them apart. The times that we spent discussing over various things - right from games, sports, movies to many more weird things are priceless and each one of them is a gem in its own right. All these discussions have immensely broadened my horizons and took me to a different plane from where I began seeking better perspective and newer knowledge levels. His incisive reasoning/understanding of things happening around have certainly left marks on me. He has changed my life beyond all imaginations.

I am also thankful to Prof. PanduRangan for his timely inputs at various points during our interactions. Turning the clock back, Prof. PanduRangan and Dr. Kannan Srinathan introduced me to the problem of Reliable communication and its cousins nearly three and half years ago. The article (Unconditionally Reliable Message Transmission over Directed Networks) in ACM-SIAM SODA 2008 has given me a new direction and did a world of good to me – from then there was no looking back. It basically transformed my mindset - one which had mere economics to one which seeks joy of research.

I also take this opportunity to thank IIIT-Hyderabad for providing an opportunity to see the world of research so closely. My research mates Anuj Gupta, Piyush Bansal, Pranav Kumar Vasishta and G. Bhavani Shankar deserve a special acknowledgment. I would also like to thank my seniors Ritesh Kumar Tiwari and Ananda Swarup Das. My lab-mates were very kind and helpful during my association with CSTAR. They include - V. Sai Satyanarayana, Abhinav Mehta, G. Uma Devi, Neeraj Kumar, Rohit Ashok Khot, Sandeep Hans and Sarat Adepalli.

This acknowledgment would be incomplete without mention of my batch mates who made my journey memorable. This includes - Rahul Sarika, Yasovardhan and Poornima.

Above all I would like to express my gratitude towards my parents and who had, are having and will continue to have a tremendous influence on my development.

Abstract

Byzantine Agreement (BA) and reliable Point-to-point communication are two of the most important and well-studied problems in distributed computing. Informally, the challenge of BA is to maintain a coherent view of the world among non-faulty players interacting over a network with few players (possibly with malicious intent) trying to disrupt the entire process. In a synchronous network over n nodes of which up to any t are corrupted by a Byzantine adversary, BA is possible only if all pair point-to-point reliable communication is possible [Dol82, DDWY93]. Specifically, in the standard unauthenticated model, $(2t + 1)$-connectivity is necessary whereas in the authenticated setting $(t+1)$-connectivity is required. Thus, a folklore is that maintaining *global* consistency is at least as hard as the problem of all pair *point-to-point* communication. Equivalently, it is widely believed that protocols for BA over incomplete graphs exist only if it is possible to simulate an overlay-ed complete graph. Surprisingly, we show that the folklore is far from true — achieving global consistency can be *strictly* easier than all-pair point-to-point communication.

In the authenticated model, it is assumed that the adversary can forge the signatures of only those nodes under its control. In contrast, the unauthenticated model assumes that the adversary can forge the signatures of all the nodes (that is, secure signatures are not used). We initiate a study on the entire gamut of BA's in between, viz., the adversary can forge the signatures of up to any k nodes apart from the up to t nodes that it can actively corrupt. We completely characterize the possibility of BA across the spectrum. Thus, our work attempts to unify the extant literature on agreement. It is, however, more than a mere attempt towards unification as it provides insights into the field. Specifically, apart from the extremes (of $k = 0$ and $k = n - t$ where aforementioned folklore is known to hold), for every intermediate k, there are several networks over which BA is possible but all-pair *point-to-point communication* is not.

Contents

List of Figures

Chapter 1

Introduction

Byzantine agreement (BA) and reliable communication (BG) are two fundamental problems in the theory of distributed computing. The problem of arriving at an agreement (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. In many practical situations, it is necessary for a group of processes in a distributed system to agree on some issue, despite the presence of some faulty processes. More precisely, a protocol among a group of n processes (t of which may be faulty), each having a value, is said to achieve Byzantine agreement, if, at the end of the protocol, all honest processes agree on a value and the following conditions hold: (1) *Agreement:* All honest processes agree on the same value; (2) *Validity:* If all honest processes start with the value $v \in \{0, 1\}$, then all honest processes agree on v; (3) *Termination:* All honest processes eventually agree.

To study the phenomena of distributing data in consistent manner across a distributed environment in presence of faults, Lamport *et al.* [LSP82], also, 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 processes' mutual distrust in the network is typically modeled via a (fictitious) centralized adversary that is assumed to control/corrupt the faulty processes. In the threshold adversary model, a fixed upper bound t is set for the number of faulty processes. 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 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*.

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 in algorithms for BA. This augmented model is called the *authenticated model* and the problem of BA in this model are referred as *authenticated Byzantine agreement (ABA)*. Subsequently, it well known that algorithms for ABA can tolerate any number of faults, which is a huge improvement over the fault tolerance bounds of (unauthenticated) BA. Owing to this vast improvement, authenticated setting is an important model in the area of fault tolerant distributed computing. In the age of modern cryptography, it is reasonable to assume availability of Public Key Infrastructure (PKI) and digital signatures over any communication network.

1.1 Our Pursuit

Though in the age of modern cryptography PKI and digital signatures are reasonable wrappers to assume over a network, they are usually based on the conjectured hardness of some problems like integer factorization [RSA78], discrete logarithms [Gam85], permutations [Sha94], lattice based problems [GPV08, Reg04] etc. Further, proofs of the hardness of these problems appear to be beyond the reach of contemporary mathematics. An elegant method to (partially) deal with such a scenario is the concept of *robust combiners* [MPW07, HKN$^+$05, MP06]. Informally, a (k, n)-robust combiner is a construction that takes in n candidate schemes for a functionality and combines them into one scheme such that if no more than k of the candidate schemes are incorrect then the combined scheme is guaranteed to correctly implement the functionality. Note that different sets of up to k candidate schemes may fail during different executions/inputs. For instance, we have a $((1, 3))$-combiner for Oblivious Transfer(OT), that is - Given any 3 protocols for OT combiner as an input, a $(1, 3)$-combiner is a blackbox construction using these 3 protocols such that the ouput is a valid OT protocol when at most one of the three given input protocols is not an OT protocol in any execution.

Combiners are excellent toolkits to have at our disposal. For instance, consider the following scenario - we do not have a single protocol to encompass all the problem space. Per se, The protocols fail over an one-third scenario's which are mutually non-intersecting. Thus, we have three protocol that cover all the possible cases exhaustively but no single protocol works for cases - implying that given any specific instance of a problem – it is obvious that the problem belongs to one of the three mutually exclusive spaces – and thus only one of these protocols may *fail* where as the other two don't. The beauty in the construction of combiners lies in their blackbox nature - we need not worry about which

of these three protocols might fail. We have a robust blackbox construction which always gives you a valid output as long as any two inputs of them implement the functionality as per the requirements.

Analogously, in the context of authenticated Byzantine agreement (ABA), 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). It is possible that the signatures schemes of up to k players may malfunction during the execution of BA protocol. Notice that different sets of players may have their signature schemes rendered insecure in different executions. Also observe that, the fact that the signature schemes of some players may become insecure is out of the hands of the players involved. For instance, think of scenario's like the scheme used by the player is already used by the adversary or the scheme chosen is not a hard one to crack or adversary luckily gets access to the private key of the other player. Hereafter, k is referred as the *robustness parameter of authentication* for a BA protocol.

Traditionally, faults in a distributed system via a fictitious entity called *adversary* (a formal modeling of faults is presented in section 2.1.3). The problems of BA has been well studied for a wide variety of adversaries (a detailed literature survey appears in Chapter 3). In the theory of distributed computing, we usually model the faults using the paradigm of worst case scenario - that is, we let an adversary to choose the faults that can occur in the system so that it can benefit in a maximal fashion from the selection. We, however, set a cap on the maximum number of faults that may occur during the protocol execution. The most common and frequent types of faults include letting t players(call this set $\mathcal{A}$) which it can control at its will and make them behave in an arbitary fashion. Along the same lines, In addition to the t players chosen by the adversary to behave an arbitrary fashion by deviating from the protocol specified, we let the adversary, also, to choose k players(mentioned in the prequel, call them $\mathcal{K}$) who do not have a secure signature scheme.[1] Observe that in all that there are $(n-t)$ non-faulty players. By non-faulty players, we refer to those players who follow the protocol given to them honestly. This is because even though the adversary can forge the signatures of all the players from sets $(t+k)$ - it can control the protocols being run by t of them by making them behave in an arbitary fashion. From now on, we shall use the notation (t,k) to capture the fault-tolerance of the network. As told in the prequel - The notion of faults in the system is usually modeled with an fictitious entity *adversary* and hence you may call this a (t,k)-adversary.

In this dissertation, we set out to study the following - In the authenticated model of BA, it is assumed that the adversary can forge the signatures of only those nodes under its control (that is $k=0$). In contrast, the unauthenticated model of BA, assumes that the adversary can forge the signatures of all the nodes (that is, secure signatures are not used or $k \geq (n-t)$). We initiate a study on the entire gamut of BA's in between, viz., the adversary can forge the signatures of up to any k nodes apart from the up to t nodes that it can actively corrupt.

[1] The worst case will be when the adversary chooses $\mathcal{A}$ and $\mathcal{K}$ disjointly, that is $\mathcal{A} \cap \mathcal{K} = \Phi$.

1.2 Contributions of the Thesis

It is widely believed that BA is possible only if every (non-faulty) player can reliably communicate with every other (non-faulty) player. Indeed, *all* known characterizations for BA support the same. For instance, in the standard non-authenticated model, it is well-known that BA is possible if and only if $n > 3t$ and G is $(2t + 1)$-connected [Dol82], which in turn implies that every node can reliably communicate with every other node [DDWY93]. Analogously in the standard authenticated model, it is equally well-known that BA is possible only if G is $(t + 1)$-connected, which again implies that all node reliable communication is possible (in the authenticated model) [Sri06]. Even in unconventional models like non-threshold adversary, BA tolerating an adversary structure $\mathcal{A}$ is possible only if no two elements in $\mathcal{A}$ are a cut-set in the network and thus, entailing point-to-point reliable communication [KGSR02]. Moreover, it is quite intuitive that *global consistency* sought by BA is "harder" than the *point-to-point* communication. For instance, (a) In an asynchronous network, global consistency is impossible even in the presence of a *single fail-stop fault* [FLP85] while point-to-point reliable communication is possible tolerating up to a *minority Byzantine faults* [SAA95]. (b) In a synchronous network, there is a "gap" in their communication complexities: All consistency protocols (throughout the paper, we deal with only deterministic synchronous protocols) suffer from the bound of $\Omega(n^2)$ communication complexity [BGP89] whereas there exist constant bit protocols for point-to-point reliable communication [PCSR06]. We show that *Byzantine Agreement* can be strictly easier than maintaining *point-to-point communication*. In other words, there are several realistic scenarios where ageement/consensus is possible even if not all pairs of non-faulty players can reliably communicate with each other.

In this thesis, we give a *complete characterization for the entire gamut* between unauthenticated BA and authenticated BA. Specifically, using the parameters t and k we capture the entire range of agreements between the two conventional models of authentication and non-authentication – that is putting $k = (n - t)$ in our results leads us to the unauthenticated setting and putting $k = 0$ leads us to the results of standard authenticated setting. Therefore, in a way our results *unify* the extant literature on *unauthenticated* BA and *authenticated* BA. The impact of the unification that we provide can be understood for the following implications and implications:

1. *Global Consistency can be easier than point-to-point communication:* From Theorem 18 it is evident that for all networks with $n > (t+k)$, $(2t)$-connectivity is sufficient for agreement. Recall that if $k > 0$ then for simulating point-to-point communication $(2t+1)$-connectivity is necessary. Consider a scenario where the signature scheme being used by the Sender, say S, is malfunctioning; in such a scenario, it is well-known that if the network is not $(2t + 1)$-connected, there must exist a player P_j such that reliable message transmission from S to P_j is impossible[DDWY93]. From the above argument one can see that BA is easier than point-to-point communication.

2. *Byzantine Agreement (BA) is easier than Byzantine Generals (BG) [LSP82]:* Informally, the problem of broadcast in presence of byzantine faults is also studied under the name of BG. Note that if a protocol for BG exists, then it vacuously is also a protocol for reliable message transmission. Till this juncture, the problems of BA and BG were thought to be isotopic forms, that is, both these problems were considered

to be equivalent. However, we have shown that BA and BG are two different problems and cannot be used interchangeably as there are several networks over which BA is possible whereas BG is impossible, in spite of having an overwhelming non-faulty majority. This turns out to be an important implication of Theorem 18. *The folklore is true only at the boundary conditions:* "BA only if reliable communication" is true only at the boundaries (standard authenticated and unauthenticated models).

3. *Optimal Network Design:* Implicit in our characterization is the design of the smallest network in which global consistency is possible for the given fault-tolerance. It is counter intuitive to see the parameter k not appearing in the expression over incomplete graphs! Hence from that perspective our characterization is pleasantly different from what we obtain from a usual mixture of models so far considered in literature [AFM99].

4. *Protocol Design:* We show that *global consistency* can be strictly easier than maintaining *point-to-point communication*. In other words, there are several realistic scenarios where BA is possible even if not all pairs of non-faulty players can reliably communicate with each other. It is evident that a new approach is necessary for designing BA protocols — this is because all the known protocols for achieving BA over arbitrary graphs has typically relied heavily on their respective counterpart over a complete graph. Thus, BA protocols over arbitrary graphs have to be designed from *scratch*.

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 reliable broadcast. This is followed by an overview of some of the popular techniques in the extant literature used in proving lower bounds of fault tolerance and designing the protocols. In Chapter 4, we study the problem of BA under a new fault model. The thesis ends with a conclusion and open problems in Chapter 5.

Chapter 2

Scope of the Thesis

The general style of problem-solving in computer science typically involves modeling and defining abstract versions of the problem that are effectively (more) suitable for mathematical study. However, mathematical models of any sort are invariably relative to a (possibly large but nevertheless particular) range of applicability (or *scope*) and therefore are usually not suitable for use outside that range. In order to understand the relation (between the model and its scope) better, we review the most prominent characteristics of a typical model. A good model should act as a tool to address a class of questions about some domain of concern of the given problem. Hence, a model is usually "tuned" according to the domain of concern and the intended inquiry. For example, consider the problem of transporting some commodity between two cities in a certain country. Given two cities A and B, we consider the following two cases, one in which the domain of concern is invariant while the inquiry changes and the second in which the inquiry is unchanged while the domain of concern changes. Specifically, restricting the domain of concern to roadways (within the transportation problem) we ask (a) "is B reachable from A?" and (b) "what is the shortest distance between A and B?". Since the first inquiry is about "possibility" a simple undirected graph model representing the map of the country's roadways is sufficient. The second enquiry requires a more sophisticated model like an edge-weighted graph. Similarly, the same question of "is B reachable from A?" may require different models for different domains of concern; for instance, while a simple undirected graph model is enough in case of roadways alone, if the domain of concern involves both roadways and railways an edge-colored multi-graph model is needed. Thus, it is clear that the model is strongly influenced by both the domain of concern and the inquiry.

Furthermore, models invariably achieve their goal by making idealizing assumptions (that simplify the task) relevant to the domain of concern and to the intended class of questions. These assumptions often restrict the range of applicability of the model. For example, in the models described in the previous paragraph, all information regarding the geometry (shape, curvature etc.) of the roads involved is completely lost since we modeled them as graphs! Therefore, questions regarding the road-contours are beyond the scope of these models.

Thus, the scope of any solution to the given problem within a specific model depends on (or, as assumed in this work, is determined by):

(a) the domain of concern within the problem,

(b) the class of questions addressed, and

(c) the idealizing assumptions made in that model.

Likewise, we now determine the scope of this thesis by fixing our domain of concern (within the problem of secure distributed In this chapter we construct a detailed and mathematically rigorous model for the problem of Byzantine Agreement. 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 Byzantine Agreement

Given a *network* of interconnected computing and routing nodes called players, each of which starts with a boolean value, each of which is augmented with addition power of *authentication*, is it possible to design a *protocol* to let the players agree on a value even if a subset of players are *faulty* and the signature schemes of some arbitary players *malfunction*?

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

2.1.1 Network Model

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) and (ii).

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, arbitarily 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, Can01, Lin03], 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 in distributed protocols(the reader may see [Can01] for further details).

Thus, a set of n players, $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ 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 distributed 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 Fault Model

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* behavior.

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 Byzantine Agreement (section 2.1): *a Byzantine Agreement protocol is a mechanism where in the attacker controlling the flow of information is the system via some subset of points cannot prevent consensus (coherent view) among the non-faulty players in the system.* 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 reliable 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 behavior:** The behavior 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 behavior 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 behaviors.

   (e) *Covert:* Formalized by Aumann and Lindell [AL07], this tries to model adversarial behavior 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 Malfunctioning of Signatures

Digital Signatures are usually based on the conjectured hardness of some problems like integer factorization [RSA78], discrete logarithms [Gam85], permutations [Sha94], lattice based problems [GPV08, Reg04] etc. Further, proofs of the hardness of these problems appear to be beyond the reach of contemporary mathematics. An elegant method to (partially) deal with such a scenario is the concept of *robust combiners* [MPW07, HKN$^+$05, MP06]. Informally, a (k, n)-robust combiner is a construction that takes in n candidate schemes for a functionality and combines them into one scheme such that if no more than k of the candidate schemes are incorrect then the combined scheme is guaranteed to correctly implement the functionality. Note that different sets of up to k candidate schemes may fail during different executions/inputs. For instance, we have a $((1, 3))$-combiner for Oblivious Transfer(OT), that is - Given any 3 protocols for OT combiner as an input, a $(1, 3)$-combiner is a blackbox construction using these 3 protocols such that the ouput is a valid OT protocol when at most one of the three given input protocols is not an OT protocol in any execution.

Combiners are excellent toolkits to have at our disposal. For instance, consider the following scenario - we do not have a single protocol to encompass all the problem space. Per se, The protocols fail over an one-third scenario's which are mutually non-intersecting. Thus, we have three protocol that cover all the possible cases exhaustively but no single protocol works for cases - implying that given any specific instance of a problem – it is obvious that the problem belongs to one of the three mutually exclusive spaces – and thus only one of these protocols may *fail* where as the other two don't. The beauty in the construction of combiners lies in their blackbox nature - we need not worry about which of these three protocols might fail. We have a robust blackbox construction which always gives you a valid output as long as any two inputs of them implement the functionality as per the requirements.

Analogously, in the context of authenticated Byzantine agreement, 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). That is, out of the n signature schemes being used by the players, at most k of them mau become insecure/forgeable during the protocol execution. Traditionally, in the literature of Distributed Computing, faults are modelled using the paradigm of worst case scenario. So, we let the adversary to choose k players, whose signature schemes may malfunction (become insecure) during the execution of the protocol in addition to the t players - whom it can control in an arbitary fashion. In other words, the adversary can forge the signatures of these k palyers. Hereafter, k is referred as the *robustness parameter of authentication* for a BA protocol. Rather this parameter can also be treated as additional faults occuring in the system along with the traditional Byzantine faults. From now on, we use the parameters (t, k) as a measure for fault tolerance. Notice, that in all there are $(n-t)$ non-faulty players.

By saying that the players are non-faulty we mean they send, receive and compute messages according to the protocol given to them.

2.1.5 Authentication Model

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, V(vk,m,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 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 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

We are now left with the task listing our assumptions which in conjunction with the prequel would explicitly determine the scope of our results. The following list (abridged to those that are generic and valid throughout the thesis; specific assumptions that are made within some sections/subsections are clearly mentioned as and when necessary):

- We assume that the *topology* of the network is *known* to each node in the network.
- We assume all the physical processes to be classical. Modelling processes as Quantum mechanical phenomenona is beyond the scope of this work.
- 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.
- 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].
- At the end of preprocessing phase, each player get his private signature keys, in addition to $n-1$ verification keys.
- Every player is fully aware of the player set taking part in the protocol.
- A player can authenticate his messages using his private key only.

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 Agreement

The problem of consensus/agreement over a point to point network, in presence of faults is one of the most widely studied problems in theory of distributed computing. 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. 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 a consensus protocol. Such protocols require to consistently distribute some data across all the players and hence rely on a consensus protocol.

We now formally introduce two very popular problems in the area of distributed computing, namely Byzantine Agreement(BA) and Authenticated Byzantine Agreement(ABA).

3.2 Byzantine Agreement (BA)

The problem was first introduced by Pease *et al.* [PSL80] and can formally be defined as:

Definition 1 (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.*

After the seminal papers by Lamport, Shostak and Pease [PSL80, LSP82], the problem of 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.

3.3 BA: Literature Survey

3.3.1 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 BA 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. Fischer and Lynch [MN82] proved that any deterministic protocol for BA will take atleast $t + 1$ rounds. The protocol given by Pease el at. [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 [Rab83] 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 BA 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 3.5.

3.3.2 Asynchronous networks

In asynchronous setting, the problem of BA was taken up by 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 3.5.

3.3.3 Popular variants

We now give a brief overview of some the popular variants of the Byzantine Agreement(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 Byzantine Agreement: [Lam83] introduced the problem of *weak byzantine agreememt* where the validity rule is modified to a weaker condition.

Definition 2 (Weak Agreement (WBA)) *Given a set of n players $\mathbb{P}$=$\{p_1$,$p_2 \dots p_n\}$ 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 BA 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 BA 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 BA. It is beyond 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].

3.4 Authenticated Byzantine Agreement (ABA)

The bound of $n > 3t$ for 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 thrawt the challenge posed by Byzantine players. The augmented problem is popularly known as *authenticated Byzantine agreement* (ABA). The problem definition of ABA is same as that of BA, 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 3 (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.*

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 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 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, ABA are two very famous problems in the area of distributed algorithms. We now present a brief literature survey on the same.

3.5 ABA: Literature Survey

3.5.1 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 BA with fail-stop faults and 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 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 ABA 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 ABA, 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 ABA. 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 ABA 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 ABA 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 ABA which compose in parallel for any number of concurrent executions can be designed tolerating $t < n$ faults.

Computational Security

Rabin [Rab83] 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 [Rab83, Tou84] require some additional data to be set up among the players (once for a life time).

3.5.2 Asynchronous network

[BT85, Tou84] formally proved a lower bound of $n > 3t$. Rabin [Rab83] 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.

3.6 Illustrations

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 and their variants. We present only those techniques that are used in further chapters.

3.6.1 Protocol Design: Snapshots from Past

We now elaborate on some of the popular techniques prevalent in the literature for designing protocols for the problems of 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 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 BA [Definition 4] tolerating t fail-stop failures over a completely connected synchronous network of n players. We remark that it is well know

that BA 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 4 (BA for fail-stop failures) *Given a set of n players $\mathbb{P}$=$\{p_1,p_2 \dots p_n\}$ and a finite domain V, $V = \{0,1\}$. Every player i, $i \in \mathbb{P}$, holds an input value $x_i \in V$ and at the end of the protocol every player i decides on a value $y_i \in V$. A protocol η among $\mathbb{P}$ solves BA 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 all players start 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. Players send their value to every player including itself. Every player p_i adds the value that it receives from player j to W_j. For, $t+1$ rounds, each player p_i sends W, $W = \bigcup W_x$, to every other player, then adds all the values to W 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 3.1

FloodSet Protocol

Player i sends 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 3.1: 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 2 *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 3 *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 4 *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 2 implies $W_i(r) = W_j(r)$, for all P_i, P_j that are active after r rounds. Lemma 3 implies that $W_i(t+1) = W_j(t+1)$, for all P_i, P_j that are active after $t+1$ rounds. ∎

Lemma 5 *FloodSet solves BA for stopping failures.*

`Proof:` Termination is obvious, by decision rule. For validity, let the players 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 3.1, $W_i(t+1) = W_j(t+1)$. 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. 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 $l = t+1$, $t+1$ 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 $\{\{1\ldots\mathrm{n}\}$ - $\{1\ldots\mathrm{k}\}\}$. As example consider the EIG tree shown if Figure 3.2.

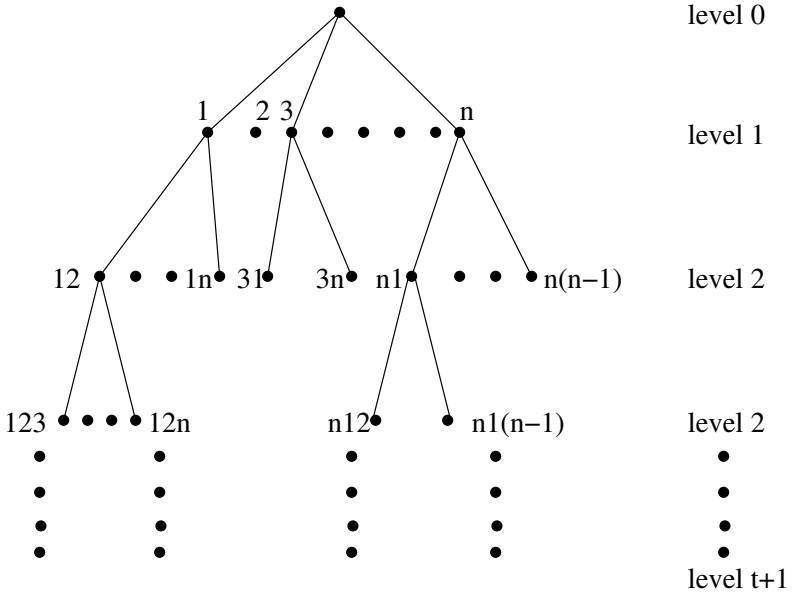

Figure 3.2: *EIG tree* $T_{n,t}$

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 BA. The first protocol, *EIGStop* [Lyn96, Page110] solves BA [Definition 4] over a completely connected synchronous graph tolerating t fail-stop faults. The second protocol, *EIGByz* [Lyn96, Page119] solves BA [Definition 4] 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 3.3. As a prelude to proving the correctness of the protocol we make following observations:

Observation 6 *After $t+1$ rounds of EIGStop algorithm, the following holds:*

1. *$val(\lambda)_i$ is player p_i's input value.*

EIG*Stop* Algorithm

Every player starts with 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 3.3: *EIGStop* algorithm

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.

Lemma 7 *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 6.2. 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 8 *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 7 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 7 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 9 *EIGStop solves BA for stopping failures [Definition 4].*

Proof: Termination follows from Lemma 8 and decision rule. For validity, If all non-faulty players start with value v, then from Lemma 8 only values that can ever decorate a player EIG tree is v and *null*. Each W_i is guaranteed to be nonempty 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 3.4.

We prove the correctness of the protocol using Lemmas 10 - 16.

Lemma 10 *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. ∎

***EIGByz* Algorithm**

Every player starts with an 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 3.4: *EIGByz* algorithm

Lemma 11 *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 10, 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 10, 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 12 *If all non-faulty players start with initial value $v \in V$, then all honest players decide on v.*

Proof: If 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 11, $val(j)_i = v$ 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 [Definition 1], 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 11 implies that if p_i is honest, then for every x, xi is a common node.

Lemma 13 *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 14 *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 15 *After $t+1$ rounds of* EIGByz*, the root node λ is common.*

Proof: From Lemma 13 and 14.

Lemma 16 EIGByz *solves BA as per definition 1 for n players tolerating up to t Byzantine faults, if $n > 3t$.*

Proof: Termination is obvious. Validity follows from Lemma 12. Agreement follows from Lemma 15 and decision rule. ∎

3.6.2 Impossibilities: Recap

"There does not exists any protocol among a set of n players, over a fully connected synchronous graph that solves BA 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 BA under this setting. But how does one prove such a statement ? Just because all known techniques of designing a protocol fails, one obviously feels so but then this cannot constitute a proof for the same. Classically, the impossibility was shown by proving that no matter what and how much information players share among themselves, one or more properties or conditions of a correct protocol will always be violated. A popular technique in the literature is *bivalency* 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 BA [definition 1] 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 BA 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 1. This contradicts our assumption of existence of Π.

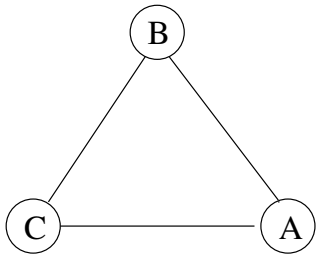

Figure 3.5: Network $\mathcal{N}$ with n=3.

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, B and C start with input value 0. In α_2, 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, A and B start with input value 1. We now argue that there exists an adversary which can ensure the following : player C can never 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 3.6.

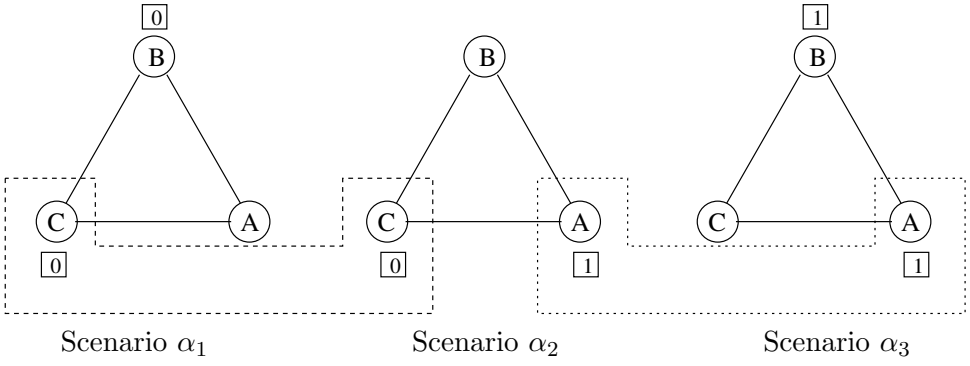

Figure 3.6: C cannot distinguish between α_1 and α_2. Similarly, A cannot distinguish between α_2 and α_3.

As per Definition 1 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 BA, 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 1. 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 behavior 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 behavior. 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 17) the impossibility of BA [definition 1] over a completely connected synchronous network $\mathcal{N}$(as shown in figure 3.5) 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 17 *There does not exists any deterministic protocol that solves BA 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 BA among 3 players over $\mathcal{N}$ tolerating 1-adversary. Using two independent copies of Π, we construct a hexagonal system S as shown in Figure 3.7. The proof proceeds by showing that S exhibits contradictory behavior. 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 neighbors 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 neighbor only by its local name which may be a.

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, B and C start with input value 0. In α_2 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, A and B start 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 3.7. 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

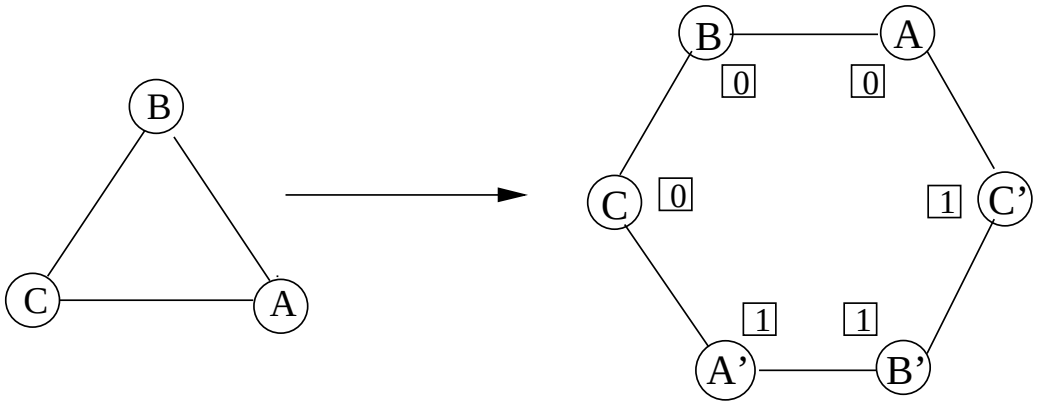

Figure 3.7: Combining two copies of Π to S

a 1-out-of-3 BA setting, therefore definition of BA [Definition 1] 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 behavior 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 3.8 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 $\alpha \overset{C}{\sim} \alpha_2$, $\alpha \overset{A'}{\sim} \alpha_2$.

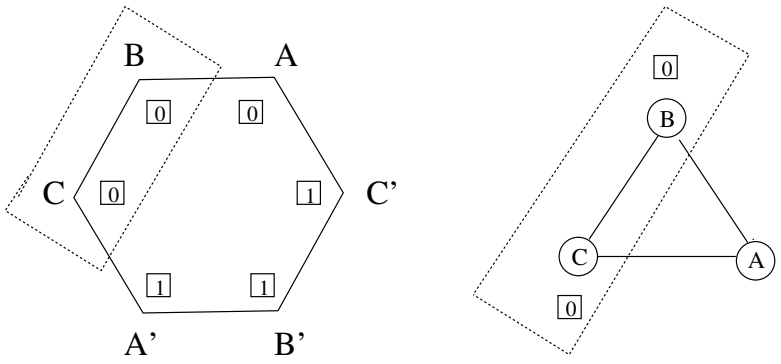

Figure 3.8: Players B, C cannot ever distinguish between α and α_1

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 C is honest and starts with value 0, as per definition of BA

[definition 1], 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 BA [definition 1], 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. ■

Chapter 4

Byzantine Agreement

4.1 Definitions

Definition 5 (Degree of a Node) *The degree of a node u, $u \in \mathbb{P}$, say δ, is the number of nodes adjacent to u. Mathematically, $\delta(u) = |\{v|(u,v)$ where u, v have an edge between them $\}|$.*

Definition 6 (Degree of a Network) *The Degree of a network $\mathcal{N}$, $\mathcal{N} = (\mathbb{P}, \mathcal{E})$, say Δ, is the minimum neighbourhood of any node in the network. Mathematically, $\Delta(\mathcal{N}) = \{n|n \leq \delta(u)$ and $u \in \mathbb{P}\}$ or $\Delta(\mathcal{N}) = Min\ \delta(u)$.*

Definition 7 (Connected Network) *A network $\mathcal{N}$ is said to be connected if there exists a path between any vertices u, v, $u, v \in \mathbb{P}$.*

Definition 8 (k-Vertex Connectivity) *A network $\mathcal{N}$, $\mathcal{N} = (\mathbb{P}, \mathcal{E})$, is said to be k-vertex connected if upon removing any set $\mathbb{A}$ such that $|\mathcal{A}| \leq k$, the subgraph induced by $\mathbb{P} \setminus \mathbb{A}$ on $\mathcal{N}$ is connected.*

Definition 9 ((2t,t)-Super-Connectivity) *A network $\mathcal{N}$, $\mathcal{N} = (\mathbb{P}, \mathcal{E})$, is said to be $(2t, t)$–Super-Connected if $\mathcal{N}$ is $(t+1)$-connected and $\Delta(\mathcal{N}) \geq 2t$.*

Definition 10 ((t,k)-BA Protocol) *Let $\mathcal{N} = (\mathbb{P}, \mathcal{E})$ be a network under the influence of a Byzantine adversary that may corrupt up to any t players and robustness parameter of authentication is k (that is, signatures schemes of up to k players may malfunction during the execution and thereby entailing their message authentication powers useless). We define a protocol Π, over $\mathcal{N}$, as a (t, k)-BA protocol if in the setting given above meets the specifications of BA given below:*

- Agreement: *All non-faulty players decide on the same value $u \in V$.*
- Validity: *If all non-faulty players start with the same initial value $v \in V$, then $u = v$.*
- Termination: *All non-faulty players eventually decide.*

Note: The problem of BA is well-defined only when $n > 2t$ and hence, throughout the paper we assume $n > 2t$.

4.2 Complete Characterization of BA

Theorem 18 (Main Theorem) *(t,k)-BA Protocol over a κ-vertex connected network $\mathcal{N}$ exists if and only if $n > 2t+ min(t,k)$ and*

$$\kappa \geq \begin{cases} t+1 & if \quad n > (2t+k) \\ (2t,t) - \textit{Super-Connected} & if \quad (t+k) < n \leq (2t+k) \\ 2t+1 & if \quad n \leq (t+k) \end{cases}$$

Organization of proof: We prove the sufficiency and necessity of κ-connectivity for $\mathcal{N}$ over Lemmas 19, 31, 32. We then prove the sufficiency and necessity of $n > 2t+ \min(t,k)$ in Theorem 41.

4.3 BA Over Incomplete Graphs

w.l.o.g. we assume that either $n > (2t+k)$ or $(t+k) < n \leq (2t+k)$ or $n \leq (t+k)$.

Lemma 19 *(t,k)-BA protocol over any general network $\mathcal{N} = (\mathbb{P}, \mathcal{E})$, $n > (2t+k)$, exists if and only if $\mathcal{N}$ is $(t+1)$-connected.*

`Proof:` *Necessity:* The necessity of $(t+1)$-connectivity is obvious. *Sufficiency:* Players run the flood-set protocol given in Figure 4.1 [1]. It is assumed that the players sign whenever they send any message and also discard any message which does not have a valid signature on it. The proof correctness is given below. ∎

Lemma 20 *All honest parties agree on whether player i's execution was clean or dirty.*

`Proof:` We say that a player sees a contradiction if at some point in the protocol they receive at least two valid messages with different content. If in any round $i \in \{1, ..., (n-1)\}$ some honest player has seen a contradiction, then by the protocol specification, all honest players see a contradiction in round $i+1$, all agree that the run was dirty, and all output a default value. We now assume that no honest player has seen a contradiction before the beginning of round n. To see a contradiction in round n, a player must receive two different messages, each with n signatures. Recall, however, that the adversary can forge at most $n-1$ signatures, since $n > t+k$, and therefore some honest player having a secure signature scheme must have signed both messages in the previous round. This contradicts our assumption that no honest player saw a contradiction prior to round n. Clearly if no honest players ever see a contradiction, then all agree that the execution was clean, and all output the same value. ∎

Lemma 21 *If the player is honest and has a secure signature scheme, then all honest parties agree on the players input and the execution is clean.*

[1] This protocol is essentially the Dolev-Strong protocol [DS83] followed by n rounds of flooding.

Flood-Set Protocol

Input: Player J start with an input bit $\sigma \in \{0, 1\}$.

Computation: Player J initiates protocol with input $\sigma \in \{0, 1\}$.

1. (Round $r = 0$) J sends (σ, Sign(skJ, σ)) to every player.
2. In round $r = 1$ to n:
   (a) Player i, $i \in \mathbb{P}$, checks every incoming message and discards any that are not (, r)-valid or that already contain i's signature. i then orders the remaining messages (lexicographically).
      i. If the content, v, of all remaining messages is identical, i appends its signature to the first message forming a (v, $r + 1$)-valid message and sends this to all neighbours.
      ii. If there exist 2 messages with different content, i appends its signature to the first 2 such messages and sends both to all players.

Player i, $i \in \mathbb{P}$, deems the execution/invocation of the flood set protocol by J as *dirty* if he detects the influence of a byzantine player (i.e., if i receives valid signatures by J on two different messages, or never receive any messages with valid signatures by J); otherwise we say that the execution is *clean* for i. (Note that in our setting an execution can be *dirty* if J has an insecure signature scheme.) This ends the flood-set protocol.

In parallel, each player i invokes the flood-set protocol (given above) with his input bit.
At the end, every player creates a n-tuple Ω. The i^{th} location of this tuple contains a $\perp$ if i's flood-set execution is *dirty*, else if it is *clean* and has a consistent, say v, i^{th} location in Ω has v. We let χ to be the number of players who have a *clean* run.

Output: Take a majority over Ω and output it; otherwise, all players output a default value.

Figure 4.1: A Flood-Set Protocol.

Proof: Since j is a non-faulty player, whose signature cannot be forged by an adversary, and the network is $(t+1)$-connected it simply implies that j's location in Ω is consistent across all non-faulty i. ∎

Lemma 22 *Flood-Set Protocol given in Figure 4.1 is a (t,k)-BA protocol, given $n > (2t+k)$ and the network is $(t+1)$-connected.*

Proof: Termination is obvious. For validity, since $n > 2t+k$ this implies $(n-t-k) > t$. Notice that the $(n-t-k)$ denotes the number of non-faulty players with a secure signature scheme. And hence we can conclude that the non-faulty players out number the corrupt players. Since, all non-faulty players start with the same value v, from Lemma 21, the decision rule simply implies that v is the only possible decision. For agreement, Lemma 20 leads us to the fact that all for any two non-faulty players i, j Ω is consistent across both of them. Same decision rule applied across all players implies agreement. ∎

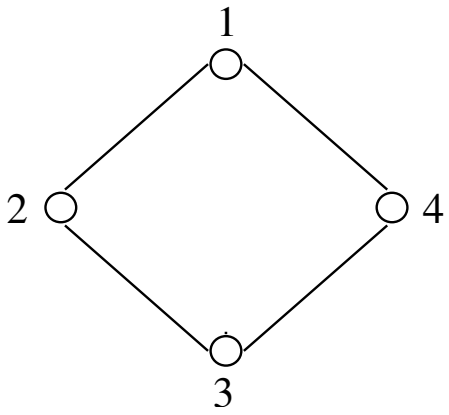

Figure 4.2: Network G

Lemma 23 *BA is possible over network G in Figure 4.2 in which either 1 or 3 is corrupt by the adversary while it is publicly known that 2 and 4 have no authentication schemes.*

Proof: Protocol is given in table 4.1 and its proof of correctness is given in Theorem 26. It is assumed that the players sign whenever they send any message and also discard any message which does not have a valid signature on it. ∎

Lemma 24 *If player 4 does not receive the input value of player 2 by the end of this protocol, player 2 can identify the Byzantine faulty player.*

Proof : *w.l.o.g.* Let player 2 starts with a input bit $\alpha \in \{0,1\}$ and let player 4 receive values α, β[2] through the two vertex disjoint paths. Now, β can take the values either $\alpha, \overline{\alpha}$ [3] or a *null*. In case β is $\overline{\alpha}$, then player 4 does not receive player 2's input. However, as per protocol player 4 now sends the value $\overline{\alpha}$ via the other path, which is non-faulty and thus,

[2]Note that since of the two paths is non-faulty, at least one of the values that player 4 receives is α.

[3]we use the notation $\overline{\alpha}$ to denote the complement of input value of player 2.

Code for player 1:

Let player 1 start with an input $\sigma \in \{0,1\}$.

1. Receives the values sent by players 2 and 4 and them across to 4 and 2 respectively.
2. Do nothing.
3. Receive the value ψ from player 4 and send it to 2.
4. Do nothing.
5. Receive the value from 2 and output the same.

Code for player 3:

Let player 3 start with an input $\sigma \in \{0,1\}$.

1. Receives the values sent by players 2 and 4 and them across to 4 and 2 respectively.
2. Do nothing.
3. Receive the value ψ from player 4 and send it to 2.
4. Do nothing.
5. Receive the value from 2 and output the same.

Code for player 2:

Let player 2 start with an input $\sigma \in \{0,1\}$.

1. Sends σ to players 1 and 3.
2. Receive values, say ψ_1 and ψ_3, from players 1 and 3 respectively.
3. Do nothing.
4. Receive the values ψ'_{31} from player 1 and similarly ψ'_{13} from player 2. If any of these values is a $\perp$ then replace it with his input value, σ.
   If $\psi'_{13} = \psi'_{31} = \sigma$, then decide on σ.
   Else if $\psi_{ij} \neq \sigma$, then decide on ψ_j.
5. Send the value decided upon to 1 and 3.

Code for player 4:

Let player 4 start with an input $\sigma \in \{0,1\}$.

1. Sends σ to players 1 and 3.
2. Receive values, say ψ_1 and ψ_3, from players 1 and 3 respectively.
3. Send the value ψ_3 to players 1 and ψ_1 to 3.
4. Create a set W from ψ_1 and ψ_3. If $|W| = 1$ decide on that element, else output σ.
5. Do nothing.

Table 4.1: BA Protocol for Lemma 23

sends them across to 2. While, the value that is received along the other path cannot be toggled to $\overline{\alpha}$ as it has the signature of a non-faulty player which cannot be forged. Upon receiving this, player 2 can easily see which of the two players 1 or 3 is faulty by looking at the signatures on the message. In the other two cases, player 4 would have received 2's input bit. ∎

Lemma 25 *If player* 4 *receives player* 2*'s input value then player* 2 *will also have the knowledge of the same.*

`Proof:` In case, player 4 gets to know player 2's input, we will show that the adversary cannot prevent player 2 from being unaware of the same. Notice that as per the protocol, player 4 sends the values received from player 1 via player 3 and vice-versa. By arguments similar to those in Lemma 24, we can see that if player 4 received player 2's value, then at the end of protocol, player 2 will not receive a toggled value of his input. So, he either receives either $\perp$ via both paths or a $\perp$ along one of the paths. Notice that in both these cases, player 2 can always be assured that player 4 received his input. ∎

Theorem 26 *Protocol given in Table 4.1 is a BA protocol on Network G in which either* 1 *or* 3 *is corrupt by the Byzantine adversary while it is publicly known that* 2 *and* 4 *have no authentication schemes.*

`Proof:` Termination is obvious. For validity - Observe that at the end of the BA protocol in Table 4.1, players output either the input value of player 2 or 4 (both these players are non-faulty) and hence, the criteria of validity will be met. For Agreement - from Lemma 25, it easy to see that player 2 can realize whether 4 has received its input value. If 4 does not receive player 2's input - by Lemma 24, 2 can find out the adversary, say player 1, and hence player 4's input value (it needs to consider the input value of player 4 received from player 2). Once, player 2 knows what value to be agreed upon, it sends this value to players 1 and 3 and thus, agreement. ∎

Lemma 27 *Every* 2*-connected network over n nodes has a $(1, \psi)$-BA protocol, where $\psi \in [2, n-2]$.*

`Proof:` We now prove that if the network $\mathcal{N}$ is 2-connected, then a $(1, \psi)$-BA protocol exists, where $\psi \in [2, n-2]$. The proof outline is as follows: we design a protocol by assuming that the adversary *always* uses his full power and corrupts exactly 1 node actively. Besides this, we also assume that $(n-2)$ players lose their power of authentication. It is straightforward to see that such a protocol would also be a $(1, \psi)$-BA protocol, where $\psi \in [2, n-2]$.

Designing the protocol Π*:* Players exchange messages as per the *flood-set protocol* given in Figure 4.1. But, now apply a "modified" decision rule, which is as follows: If a majority exists over all the *clean* runs, then he outputs that value as his *decision* and halts. Else, he does the following: If the number of *clean* runs is more than 2, then he outputs a *default* value and halts. However, if the number of *clean* runs is only 2, say, only runs of players a and b in Ω were *clean*. Notice that, one of the players a or b must be non-faulty and having a secure signature scheme while the other player may be corrupt or non-faulty.

We now make a big assumption: We let both a and b to be non-faulty. Every player $k \in \mathbb{P} \setminus \{a, b\}$ routes/sends his value to player a in the following priorities: (a) through the

direct edge(if it exists), (b) otherwise if a and b are *not* adjacent then a, b have at least 2 vertex disjoint paths between them and all the nodes in these paths must use these paths only. (c) else let $\chi = \{p_1, p_2..., p_j\}$ which is the set of paths to a, if any such path includes b route it along it else choose one in lexicographic ordering (say). Similarly, he also sends it to b. Players a and b are required to sign and send them back to player k. If a and b receive more than one value from a player or not along the routing protocol given, a and b will not respond. Suppose, k receives, say, α and β from a and b respectively. If any of the same do not match his input bit, he drops that message (Note that he cannot infer anything whether the player who signed on the toggled bit is corrupt or not).

After all the players receive the messages signed by a and b, each player runs the *flood-set protocol* given in Figure 4.1 once with its input bit signed by a and the next time signed by b. If both these runs turn out to be *clean* but the values contradict each other – the player's run is overruled to be *dirty*. And if any one of these two runs is *clean* – the player's run is ceded to be *clean*. At the end of *Flood-Set protocol*, players take a majority among all clean runs including the two earlier runs of a and b; otherwise decides on a default value and the protocol terminates. This extension would work as along as *one* of the player's execution in the flood-set protocol is *clean*. We, now, prove that is rather an invariant if both a and b are non-faulty.

Claim 28 *If both a and b are non-faulty then at least one of the runs are clean.*

Reason: Since $\mathcal{N}$ is 2-connected - two cases arise: (a) a, b are a vertex cut-set[4] in $\mathcal{N}$. (b) a, b are not a vertex cut-set. In the former, it is easy to see that the claim is maintained as there shall be at-least two components upon removing a and b and it is clear that the adversary may be present only in one component. Thereby, the players in the other component can get the signature of a and b and hence, they will be able to sent their value to all other nodes successfully - one of a or b is non-faulty with a secure signature scheme and the other is non-faulty - so either both the runs will be clean with a consistent value or one of the runs will be clean and the other will be dirty - By definition, both these cases are deemed to be *clean* runs. The case of both runs being clean with a contradictory value can happen only if one of a, b and the player is faulty.

In the later, there is only one component upon removing a and b. If a and b are adjacent, notice that the both a as well as b are sure to have a neighbors which is other than b and a respectively (as $\mathcal{N}$ is 2-connected, neighborhood of each node is at least 2). In this case, one of them is guaranteed to be non-faulty and hence from the routing method it is easy to see that, one of these node's will have a *clean* run (arguments go similar to previous case). If a and b are neither vertex cut-sets nor adjacent then there are at least 2 vertex disjoint paths between a and b. And active adversary resides in only one of them. Hence, at least of the nodes in the other path will send its input bit to get signed from a, b as per the routing algorithm. Hence, it easy to see that such a node *always* exists and hence at least one of the nodes will have a *clean* run. And, thus the claim.

Observe that, all the players would have agreed on a value, under the big assumption that both a and b were non-faulty. However, if the protocol has not terminated yet - it is implied that either a or b is faulty (for otherwise, from the above claim players would

[4] A vertex cut-set in a graph is a set of vertices whose removal from the graph makes it disconnected.

have decided already). Depending on whether players a and b are a vertex cut-set or not, players($\mathbb{P}$) do the following:

Case 1: If a and b are *not* a vertex cut-set, a publicly chosen (say least numbered player outside a, b) non-faulty player may send his input to everyone using paths outside a and b. He sends the message to a through a path avoiding b and to b via paths avoiding a.

Case 2: If a and b are a vertex cut-set, say a and b partition $\mathcal{N}$ into x components $c_1, c_2, \ldots c_x$. Now, we choose a (public) representative from each of these components, say n_i from is chosen from c_i. We now create a virtual network $\mathcal{N}'$ on nodes n_i's, a and b in which an edge appears between any two nodes only if there is a direct edge between the components represented by them. Notice that each of the n_i's has to be connected to both a and b as if they are only connected to either to a or b only then $\mathcal{N}$ cannot be 2-connected. Thus, $\mathcal{N}'$ has to be 2-connected. Now, each node in $\mathcal{N}'$ (other than a and b) forms a pair with another node (pairings are publicly known), call these pairings $\mathcal{P}$. Say, n_x is paired with n_y. Players simulate the following *sub-protocol:* Players a, b, n_x and n_y run the steps (1) and (2) of the BA-protocol given in table 4.1 with players a and b running codes of players 1 and 3 and n_x and n_y simulate players 2 and 4 respectively while the remaining just act as routers. The only difference in this execution when compared to the protocol given in table 4.1 is that instead of step (3), player 4 uses the flood-set protocol given in figure 4.1 on network $\mathcal{N}$ with the tuple containing inputs ψ_1 and ψ_3. If it is a *clean* run, players decide on that value and halt.

Notice that if players have not agreed on any value, it means that player 2 can identify the adversary as follows: Since W is not singleton, 2 needs to look at the signatures on the value that is different from his input and whomsoever of 1 or 3's signature is the first to appear on it has to be the adversary! Here, since n_x is simulating 2, he knows the faulty player (out of a and b). If players havn't decided yet: players n_x and n_y swap their codes, that is, n_x deploys the code of 4 and vice-versa and then, they re-execute the sub-protocol given above. If players still did not agree, both n_x and n_y know who the faulty player is (out of a and b). The executions of the sub-protocol given above proceeds until players have agreed or no more pairings remain in $\mathcal{P}$. When all pairs(decided in the prequel) exhaust – Notice that all n_i's would have known who the adversary is and this as good as the adversary making himself public!

All the n_i's agree on the input of the non-faulty player (out of a and b). Now, each of these n_i's send the decision to all players in the component represented by it and also to a and b. This completes the construction of Π ($(1, n-2)$-BA protocol).

Termination is obvious. For agreement, by using the *Flood-Set protocol* (Figure 4.1) as a sub-routine we are ensuring that all the non-faulty players have consistent values and hence the decision rule simply implies that all of them *agree* on the same value. If all non-faulty players start with same input σ, then both a and b *cannot* be non-faulty. Recall that, there is only one faulty player in the execution. Hence, any other player's input has to be σ and the protocol decides only upon receiving another player's input. Thus, if all non-faulty start with the same input, σ is the only possible output. Hence, by definition, it is a $(1, n-2)$-BA protocol over $\mathcal{N}$. ∎

We intend to capture the faults by a fault structure, that is, an enumeration of all the possible "snapshots" of faults in the network. Note that a single snapshot can be described by an ordered pair (B, K), where $B, K \subseteq \mathbb{P}$ and $B \cap K = \emptyset$,[5] which means that the players in the set B are Byzantine faulty while the players in the set K may have their authentication schemes entailed useless. Thus, a fault structure is a collection of such pairs. More precisely, we define the fault structure by $\mathbb{A}$, where $\mathbb{A} \subseteq 2^{\mathbb{P}\times\mathbb{P}}$. The fault structure is *monotone* in the sense that if $(B_1, K_1) \in \mathbb{A}$, then $\forall (B_2, K_2)$ such that $B_2 \subseteq B_1$ and $K_2 \subseteq K_1$, $(B_2, K_2) \in \mathbb{A}$. We note that $\mathbb{A}$ can be uniquely represented by listing the elements in its *maximal basis* $\overline{\mathbb{A}}$ which we define below. In what follows, unless specified otherwise, we work with only the maximal basis of $\mathbb{A}$.

Definition 11 (Maximal Basis of $\mathbb{A}$) *For any monotone fault structure $\mathbb{A}$, its maximal basis $\overline{\mathbb{A}}$ is defined as $\overline{\mathbb{A}} = \{(B, K) | (B, K) \in \mathbb{A}, \nexists (X, Y) \in \mathbb{A}, (X, Y) \neq (B, K), X \supseteq B$ and $Y \supseteq K\}$*

Note that another way of representing the faults under consideration (viz., t-adversary and k is robustness parameter of authentication schemes) is via an Fault Basis $\mathbb{A}$ given in definition 11 with the understanding that the any one pair (B_i, K_i) from $\mathbb{A}$ may be the possible "snapshot" of the faults in $\mathcal{N}$ and $\mathcal{A}$ may corrupt the players in B_i and players in K_i may have a malfunctioning signature scheme. It is evident that a (t, k) faults is characterized by a fault basis $\mathbb{A} = \{(B, K) || B| \leq t, |K| \leq k, B \cap K = \emptyset\}$.

Theorem 29 *(t, k)-BA protocol over any general network $\mathcal{N} = (\mathbb{P}, \mathcal{E})$, when $(t + k) < n \leq (2t + k)$, tolerating a 3-sized fault basis $\mathbb{A} = \{(B_1, K_1), (B_2, K_2), (B_3, K_3)\}$, $\forall (B_i, K_i) \in \mathbb{A}$, $|B_i| = t$ and $|K_i| = k$, exists if and only if $\mathcal{N}$ is $(2t, t)$-Super-connected.*

`Proof:` *Sufficiency:* We now extend the protocol Π (refer Lemma 27) which is a $(1, n-2)$-BA protocol over a 2-connected network to a new protocol χ over any general network $\mathcal{N} = (\mathbb{P}, \mathcal{E})$ satisfying Theorem 29. It is evident that such a network $\mathcal{N}$ should satisfy the following: we are guaranteed the presence of an non-faulty player with a signature scheme that is working (Since $(t + k) < n$ and $|B_i| + |K_i| \neq |\mathbb{P}|$). Let us call the non-faulty player with a secure signature scheme to h_i when $\mathbb{A}_i$, $\mathbb{A}_i \in \mathbb{A}$, is corrupt.

Upon removing the players in $(B_1 \cup B_2 \cup B_3)$ from $\mathcal{N}$, say $\mathcal{N}$ is paritioned into x components viz. $c_1, c_2, \ldots c_x$. Now we chose a representative, say $r_j, j \in [1, x]$, from each of these components in the following order: (a). If the component has a h_i(defined in prequel), he is chosen. Note that, at most two of the three h_i's, say h_α and h_β, may lie inside a single B_i. In this case, h_α and h_β combine to create a new virtual node $r_{virtual}$ as follows: If both have a path consisting of nodes exclusively from B_i or B_i and outside the remaining two B's (other than B_i), they may use this path to simulate player $r_{virtual}$. However if all paths cut across remaining two B's, $r_{virtual}$ functions as follows: Any player i wanting to send a message to $r_{virtual}$ sends the messsage to both of them and they now send the values to each other via any two paths (chosen publicly) such that one of them avoids B_j and the other avoids B_k. h_α and h_β now take a majority over these values among the values obtained in the *clean* runs. Virtual node $r_{virtual}$ sending a message to i is same as h_α and h_β agreeing

[5]This is not a serious assumption as if there some players common to both sets, such players can *w.l.o.g* placed solely in set B.

on a value (that is, sending via the both the paths and then, taking a majority among the *clean* runs) and sending the transcripts to player i. Observe that, both (h_α and h_β) are either a part of the adversary or both of them are non-faulty with one of them is guaranteed to have a secure authentication scheme. In the creation of a virtual node - in the former it is easy to see that this property will be maintained as the path is exclusively chosen from B_i. For the latter, one of the paths is guaranteed to be non-faulty, as only one of B_i, B_j, and B_k is corrupt.

The virtual node being created has the following property: Either it is non-faulty with a secure signature or faulty. In the former, our purpose to create a non-faulty node with secure signature is realized as follows: In this case B_i is not corrupt, one of h_j or h_k has a secure signature scheme. If they have a path exclusively in B_i then it is easy to see that the construction works. When the construction uses 2 paths (one from B_j and the other from B_k), if one of h_j or h_k hassignature scheme which the adversary cannot forge - the only value that can be received consistently is the value of the honest player. While in the latter case of the virtual node being fauly, the protocol is anyways not dependent on this node.

Finally, let us say n_i is chosen from c_i (components formed after removing B_i, B_j and B_k from $\mathcal{N}$) and b_i from B_i. We now create a virtual network $\mathcal{N}'$ on n_i's and b_i's in which an edge appears between any two players(representatives) only if there is a edge between the components represented by them. Each of these n_i's is connected to at least two b_i's (Observe that $\mathcal{N}$ is $(t+1)$-connected and each node n_i has a neighborhood of atleast $2t$. Also, n_i cannot have an edge with a player in some other component c_j. If there is a direct edge between two components – For if there is an edge, these cannot belong to two different components. Hence, each n_i is connected to at least 2 of the 3 b_i's). Thus, $\mathcal{N}'$ is now a 2-connected with at most one of the three b_i's being faulty and one of the n_i's being non-faulty with a secure authentication scheme. Players in $\mathcal{N}'$ now run protocol Π (given in Lemma 27) and Lemma 27 implies that on $\mathcal{N}'$, Π is a $(1,\psi)$-BA protocol. Thus, players in $\mathcal{N}'$ (representatives) agree. After the representatives agree, they distribute the decision across their components and also to the representatives of B_i's. For all nodes inside B_i, since the network $\mathcal{N}$ is $(2t,t)$-Super-connected and hence a degree of at least $2t$, each node in B_i has an edge to a node each from B_j and B_k or a direct edge to a node outside all B_i's. In the latter, he decides on the value obtained from outside B_i's. In the former, he needs to take a majority among the three values received from each of the B_i's, which is bound to exist as two of the three B_x's are non-faulty and would have agreed in the execution of Π. The proof of correctness stems from the fact that once the representatives agree (as given in Lemma 27, it is easy that all the players in their respective components also agree. This completes sufficiency of Lemma 31.

Necessity: We shall now prove that a (t,k)-BA protocol over a network $\mathcal{N}$ on n nodes, $(t+k) < n \leq (2t+k)$, does not exist if $\mathcal{N}$ is not $(2t,t)$-Super-connected. If a network is not $(2t,t)$-Super-connected it implies that either $\mathcal{N}$ is not $(t+1)$-connected or $\mathcal{N}$ has a node, call it u, with a neighborhood of utmost $(2t-1)$ nodes. The necessity of the former($(t+1)$-connectivity) is straight forward. For the latter case, we shall go ahead and prove the impossibility over a 2-sized fault basis $\mathbb{A} = \{(B_0, \mathbb{P}\setminus(B_1 \cup B_0)),\ (B_1, \mathbb{P}\setminus(B_1 \cup B_0))\}$. We let B_0 and B_1 to be any arbitrary partitions of $N(u)$ and u, where $N(u)$ is the neighborhood of u. Define two executions $\mathbf{E}_0$ and $\mathbf{E}_1$ as follows. In the execution $\mathbf{E}_\alpha \in \{\mathbf{E}_0, \mathbf{E}_1\}$, the set

B_α is Byzantine corrupt and all nodes *except* those in $B_{\overline{\alpha}}$ may have an insecure signature scheme. u starts with input 0 in both these executions. In $\mathbf{E}_\alpha$, all the nodes in $\mathbb{P}\setminus\{B_\alpha\setminus\{u\}\}$ start with input α. The behavior of the Byzantine set B_α in the execution $\mathbf{E}_\alpha$ is to send no message whatsoever to anyone other than u and to u send exactly the same messages that are sent by an honest B_α in the execution $\mathbf{E}_{\overline{\alpha}}$. In order for the Byzantine corrupt B_α to behave as specified in the execution $\mathbf{E}_\alpha$, the adversary needs to simulate the behavior of $\mathbb{P}\setminus B_{\overline{\alpha}}$ in the execution $\mathbf{E}_{\overline{\alpha}}$. To achieve this task, the adversary simulates round-by-round the behavior of the vertices in $\mathbb{P}\setminus B_{\overline{\alpha}}$ using their respective inputs in the execution $\mathbf{E}_{\overline{\alpha}}$. At the beginning of each round, each simulated player has a history of messages that it got in the simulation of the previous rounds and signs on behalf of all the players in $\mathbb{P}\setminus B_{\overline{\alpha}}$. Notice that in $\mathbf{E}_{\overline{\alpha}}$ the set $B_{\overline{\alpha}}$ does not any send messages to anyone except u, thus in $\mathbf{E}_\alpha$, B_α need not forge the signature of nodes in $B_{\overline{\alpha}}$ and hence the simulated player sends during the simulation the same messages that the honest player would send in the original protocol in the same state. The simulated messages that (players in) B_α sends to u are really sent by the players. All other messages are used only to update the history for the next round. The messages which are added to the history of each simulated vertex are the real messages that are sent by players and the simulated messages that are sent by the vertices in B_α. The history of messages of each simulated vertex in execution $\mathbf{E}_\alpha$ is the same as the history of the vertex in execution $\mathbf{E}_{\overline{\alpha}}$. Therefore, the messages sent by B_0 and B_1 to u in both executions are exactly the same. Thus, u cannot distinguish whether it is in $\mathbf{E}_0$ or $\mathbf{E}_1$ and hence, u remains in a bivalent state.[6] Hence, adversary can ensure that some non-faulty node will remain bivalent for as long as it wants! This completes the necessity and thus, the proof of Theorem 29. ∎

We, now, extend the characterizations over a 3-sized fault basis to an n-sized fault basis.

Lemma 30 *(t,k)-BA protocol over a network $\mathcal{N}$ tolerating a fault basis $\mathbb{A}$ exists if and only if there exists (t,k)-BA protocols for every 3-sized fault basis $\mathbb{B}$, $\mathbb{B}\subseteq\mathbb{A}$.*

`Proof:` This extension is based on the works of Hirt and Maurer [HM97]. *Sufficiency:* Given a protocol ψ solving BA on a 3-sized fault basis we now show how to extend it to a protocol χ tolerating a 4-sized fault basis $\mathbb{A}'$. We divide $\mathbb{A}'$ into four parts $\mathbb{A}_1$, $\mathbb{A}_2$, $\mathbb{A}_4$ and $\mathbb{A}_4$ such that $\forall i,j\in\{1,2,3,4\}$ $\mathbb{A}_i\cap\mathbb{A}_j=\emptyset$ and $|\mathbb{A}_i|\neq 0$. Now, consider the fault basis $\mathcal{A}_i=\{\mathbb{A}\setminus\mathbb{A}_i\}$. Since, we are given ψ to solve all the 3-sized fault basis, call them ψ_1, ψ_2, ψ_3, ψ_4. It is easy to see that every element of $\mathbb{A}'$ is present in at least in three of the four sets $\mathcal{A}_i$'s. Hence, any $x\in\mathbb{A}'$ is tolerated by at least *three* of the four ψ's. Players, now, simulate a $(1,3)$-BA protocol [Lyn96], call it Π_4^1, (tolerating a single Byzantine fault in the *non-authenticated setting*) on 4 virtual players, say p_1, p_2, p_3 and p_4. The simulation proceeds as follows: All players take part in the simulation these virtual players. Players, now, execute the BA protocol Π_4^1 on these virtual players. Let π_i be the code run by the player p_i for BA protocol Π_4^1. Virtual player p_i sending message to virtual player p_j is same as all the players (part of the simulation) running ψ_j on the output of ψ_i. At the end of the simulation of Π_4^1 protocol, each player in $\mathbb{P}$ takes a majority on the decision value of these four virtual players. Note that any non-faulty player gets the same value as decisions for 3 out of 4 virtual players are guaranteed to be same (since, only one of the

[6] Nodes outside B_0 and B_1 can never distinguish if u was indeed non-faulty or was, rather, faulty. Hence, u being a part of adversary is of pivotal importance.

virtual players are faulty). This completes the construction of χ tolerating a 4-sized fault basis from protocol solving a 3-sized fault basis. Now, given a 5-sized fault basis $\mathbb{C}$, we can again create four parts $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and $\mathcal{A}_4$ as mentioned earlier. Note that each of these $\mathcal{A}_i$'s is either a 4-sized basis or a 3-sized one and we now have have constructed protocols for both 3 as well as 4 sized fault basis. And $\forall x \in \mathbb{C}$, x is tolerated by at least three $\mathcal{A}_i$'s and hence we can again run the BA protocol Π^1_4 over these four virtual players. In general, given any n-sized fault structure $\mathbb{A}$ we can use the simulation technique given above of creating 4 sub-structures($\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$ and $\mathcal{A}_4$) such that each $\forall x \in \mathbb{A} \exists i, j, k \in \{1, 2, 3\}_{i \neq j \neq k}$ and $x \in \mathcal{A}_x | \forall x \in \{i, j, k\}$ and notice that we have effectively reduced the size of $\mathbb{A}$ by at least one in each of the $\mathcal{A}_i$'s and if the size of any $\mathcal{A}_i$ is greater than 3 we recurse on it. Recursion terminates when the size of the fault basis is three and we have BA protocols for all the 3-sized fault basis. *Necessity:* In case, a protocol Π solving a n-sized fault basis such that $n > 3$ exists it is easy to see that it can as well be deployed as a protocol Π' that is required to tolerate a 3-sized fault basis. ∎

Lemma 31 *(t, k)-BA protocol over any general network $\mathcal{N} = (\mathbb{P}, \mathcal{E})$, $(t+k) < n \leq (2t+k)$, exists if and only if $\mathcal{N}$ is $(2t, t)$-Super-connected.*

`Proof:` Follows from Lemma 30. ∎

Lemma 32 *(t, k)-BA Protocol over any arbitrary network $\mathcal{N}$, $n \leq (t + k)$, exists if and only if $\mathcal{N}$ is $(2t + 1)$-connected.*

`Proof:` Since $n \leq (t+k)$, it basically means that the signatures schemes of all players outside adversary's control were malfunctioning and hence the power of authentication is entailed useless. Hence, proofs from the standard unauthenticated model of BA [Lyn96, Dol82] will lead us here. ∎

Lemmas 19, 31, 32 complete the characterization over incomplete graphs.

4.4 BA over Complete Graphs

We now present a proof for the necessity and sufficiency of (t, k)-BA Protocol over complete networks. In other words, prove the necessity and sufficiency of $n > 2t + \min(t, k)$ for the existence of a (t, k)-BA protocol over a complete network of n nodes. Our proof is based on proof technique developed by Fisher *et al.* [FLM85]. As a prelude we formulate the notion of *view* of a player in an execution of a BA protocol. Intuitively, by *view* we want 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. Let $msg^{\Omega}_i(a, b)_a$ denote the message sent by player a to player b in i^{th} round of execution Ω of some BA protocol. The subscript a represents the last player who authenticated the message. W.l.o.g we assume that players always authenticate their messages before sending.

Then view of a player a during execution Ω at the end of round i, denoted by $view^{\Omega}_{a,i}$, can be represented as collection of all the messages it ever send and receives. Formally:

$$view^{\Omega}_{a,i} = \bigcup_k (msg^{\Omega}_k(a, x)_a, msg^{\Omega}_k(x, a)_x), \ \forall k \in \{1 \ldots i\}, \ \forall x \in \mathbb{P} \tag{4.1}$$

Here $\mathbb{P}$ is the set of all the players running the protocol. The messages sent by player a in any round i of some execution say Ω depends on 4 parameters: input value with which a starts, secret key used by a for authentication, code being executed by a, and messages received by a up to round $i-1$ of Ω. Since the outgoing messages are a function of incoming messages, we can rewrite the equation 4.1 as:

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

In order to show that the views of 2 different players a, b running in 2 different executions 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 similar code [7], and if for every round $1 \dots i$ their corresponding incoming messages are same, then their views till round i will also be same. [8] Formally:

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

We first prove that there does not exist any BA protocol over a completely connected graph(G) of four nodes tolerating a fault-structure $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. Our proof starts by assuming that there exists a BA protocol ϖ over a complete graph G (Figure 4.3) tolerating fault-structure $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. Using ϖ we construct another protocol ϖ' [Definition 12] such that existence of ϖ implies existence of ϖ' (Lemma 33). Using two copies of ϖ' we construct a system S as shown in Figure 4.3. We then prove that S must exhibit a contradictory behaviour. This implies that our assumption about existence of ϖ is wrong.

Definition 12 (ϖ') *For all players $A, B, C, D \in \mathbb{P}$, Following statements are changed:*
"C sends message m to A" in ϖ is replaced by "C multicasts message m to all instances of A(i.e. A, A') [9]
"C sends message m to B" in ϖ is replaced by "C multicasts message m to all instances of B(i.e. B, B')
"D sends message m to B" in ϖ is replaced by "D multicasts message m to all instances of B(i.e. B, B')
"A sends message m to B" in ϖ is replaced by "A multicasts message m to all instances of B(i.e. B, B')
Rest all statements in ϖ' are same as ϖ.

Lemma 33 *If ϖ exists then ϖ' exists.*

Proof: Implied from Definition 12. ∎

[7] Note that a, b may even run different codes say θ and θ', however 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 $\mathcal{I}$.

[8] [FLM85] captured this via **Locality Axiom**. In BA a player may also use its private key to determine the outgoing messages. Thus both players having same secret key is must.

[9] A and A' are independent copies of a with same authentication key

Construction of S: Take two independent copies of ϖ' and construct a octagonal system S as shown in Figure 4.3. We neither know what S is nor what does it solve. All we know is that S is a synchronous system and has a well defined behaviour. That is, for any particular input assignment S must have a well defined output distribution. Further, no player in S known the complete system. Each player in S in only aware of his immediate neighbors. In reality, a player may be connected to either a or a', but it cannot differentiate between the two. It knows its neighbor only by its local name which may be a. Note that connectivity in S is not same as in G. To be precise, in-neighborhood of any node a(or a') in S is same as in-neighborhood of corresponding node a in G, however out-neighborhood of some nodes in S is not same as out-neighborhood of corresponding nodes in G. This would make a difference if players in both systems were running same protocol(ϖ). S is constructed in a such a way that whatever messages are sent to some selected players in S, same messages can be ensured by adversary to those very selected players in G.

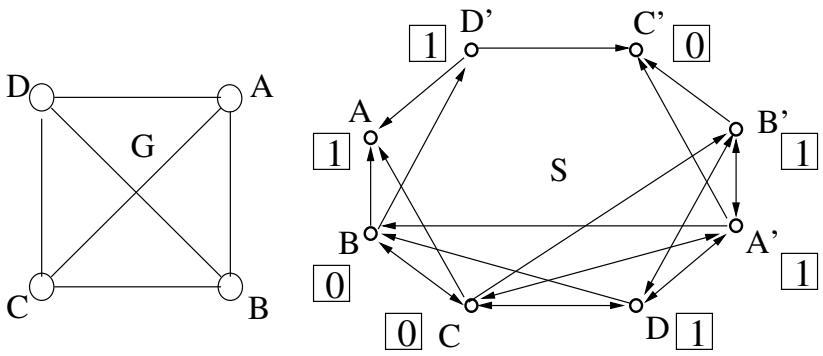

Figure 4.3: Graph G and System S.

Let β_1 be a scenario where C is an honest player. $\mathcal{A}$ corrupts A,D and B has his signature scheme broken. Here both non-faulty players B, C start with input value 0. Let β_2 be a scenario where C,D are honest players. $\mathcal{A}$ corrupts B and A has his signature scheme broken. Here player C starts with input 0 and A, D starts with input value 1. Similar, in scenario β_3 where A,D are honest players. $\mathcal{A}$ corrupts C and B has his signature scheme broken. Here all the non-faulty players (A, B, D) start with input value 1. Further, let β be an execution of S where each player starts with input value as shown in Figure 4.3. All the players in β are honest and follow the designated protocol correctly.

We now show 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 in β_2 and whatever view A',B',D get in β, $\mathcal{A}$ can generate the same view for A,B,D in β_3.

Using definition 4.3, what we essentially want to show is that adversary can ensure that for any round j of ϖ B,C in β_1 get same messages as B,C get in round j of β. Note that what players send to B, C in round j of β also depends on what these players(those who sent) receive in round $k-1$ of β. This in turn depends on what they receive in round $j-2$ and so on. Note that this continues in a recursive manner until recursion stops at round 1. The entire recursion can be visualized as trees which we refer to as execution trees T^B_β and T^C_β. Similarly, for execution β_1, executions trees $T^B_{\beta_1}$ and $T^C_{\beta_1}$, shown in Figure 4.6. We

now formally describe tree T^x_β. 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 β. All edges are directed from child to parent and are between adjacent levels only. Observe that for the proof to go through, in-degree for any node y'(or y) in system S has to be same as in-degree of corresponding node y in G. Thus structurally both trees $T^{x'}_\beta$(or T^x_β) and $T^x_{\beta_1}$ will be exactly same (A node y' in T^x_β is replaced by its corresponding node y in $T^x_{\beta_1}$). Now consider some node b'(or b) at level j in T^x_β. Then its corresponding node at level j in $T^x_{\beta_1}$ is b. Note that if the messages received by b' in T^x_β is same as those received by b in $T^x_{\beta_1}$ and both b' and b start with same input value, same private key and run same code then both will send same messages.

We now give the adversary for scenario β_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. 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 $msg^{\beta_1}_{i-1}(C,A)_C$ using A's secret key and sends it to B,D. Similarly, $\mathcal{A}$ authenticates $msg^{\beta_1}_{i-1}(C,D)_C$ using D's secret key and sends it to A,B. For $msg^{\beta_1}_{i-1}(B,A)_B$, $\mathcal{A}$ 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 $msg^{\beta_1}_{i-1}(B,A)_B$, such that $msg^{\beta_1}_{i-1}(B,A)_B \sim msg^{\beta_2}_{i-1}(B,A)_B$, authenticates it using A's key and sends it to C. If $msg^{\beta_1}_{i-1}(B,A)_B$ 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 $msg^{\beta_1}_{i-1}(B,D)_B$. 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 $msg^{\beta_1}_{i-1}(B,D)_B$ such that $msg^{\beta_1}_{i-1}(B,D)_B \sim msg^{\beta_2}_{i-1}(B,D)_B$, authenticates it using D's key and sends it to C. If $msg^{\beta_1}_{i-1}(B,D)_B$ 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 $msg^{\beta_1}_i(B,A)_A$, $msg^{\beta_1}_i(C,A)_C$ and $msg^{\beta_1}_i(D,A)_D$ via A.

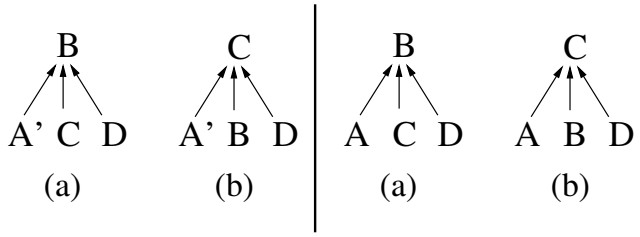

Figure 4.4: T^B_β, T^C_β and $T^B_{\beta_1}$, $T^C_{\beta_1}$ at the end of round 1.

We now argue that following the above mentioned strategy, $\mathcal{A}$ can ensure that the messages

received by B,C in round i of β_1 are same as the messages received by B,C respectively in round i of β. Formally:

Lemma 34 *$msg_i^{\beta}(x,B)_x \sim msg_i^{\beta_1}(x,B)_x$ and $msg_i^{\beta}(x,C)_x \sim msg_i^{\beta_1}(x,C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$.*

Proof: To prove that the view [Equation 4.2] of B is same in β and β_1 we apply induction on heights of T_{β}^{B} and $T_{\beta_1}^{B}$. Similarly using T_{β}^{C} and $T_{\beta_1}^{C}$, we show view of C is same in β and β_1. 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. We analyze these trees in bottom up manner. Consider trees T_{β}^{B}, T_{β}^{C} and $T_{\beta_1}^{B}$, $T_{\beta_1}^{C}$ at the end of round 1 as shown in Figure 4.4.

We now show that $\mathcal{A}$ can ensure that at the end of round 1, B receives same messages in β and β_1. Consider (a) in Figure 4.4. 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. Similarly one can show that C also gets same messages at the end of round 1 in β and β_1.

We now claim that the similarity holds for round 2 as well i.e. $msg_2^{\beta}(x,B)_x \sim msg_2^{\beta_1}(x,B)_x$ and $msg_2^{\beta}(x,C)_x \sim msg_2^{\beta_1}(x,C)_x$, $\forall x \in \mathbb{P}$. Consider trees T_{β}^{B}, T_{β}^{C} and $T_{\beta_1}^{B}$, $T_{\beta_1}^{C}$ at the end of round 2 as shown in Figure 4.5. Consider node C at level 1 in T_{β}^{B} and $T_{\beta_1}^{B}$. 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 $msg_1^{\beta}(A',C)_{A'} \sim msg_1^{\beta_1}(A,C)_A$ and $msg_1^{\beta}(D,C)_D \sim msg_1^{\beta_1}(D,C)_D$.

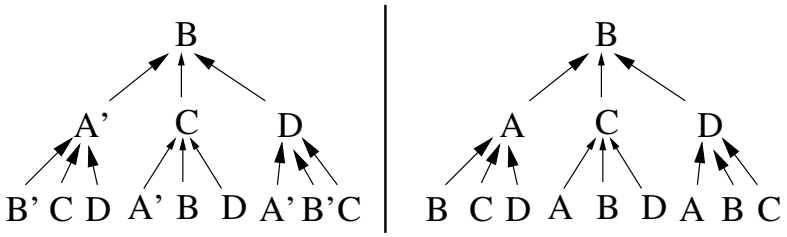

Figure 4.5: T_{β}^{B} and $T_{\beta_1}^{B}$ at the end of round 2.

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 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 has insecure signature scheme in β_1, $\mathcal{A}$ on behalf of B can construct $msg_1^{\beta_1}(B,A)_B$ such that $msg_1^{\beta}(B',A')_{B'} \sim msg_1^{\beta_1}(B,A)_B$. 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 $msg_1^{\beta}(D,A')_D \sim msg_1^{\beta_1}(D,A)_D$. 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. $msg_2^{\beta}(A', B)_{A'} \sim msg_2^{\beta_1}(A, B)_A$. Similarly one can show that $msg_2^{\beta}(D, B)_D \sim msg_2^{\beta_1}(D, B)_D$. Thus $msg_2^{\beta}(x, B)_x \sim msg_2^{\beta_1}(x, B)_x$, $\forall x \in \mathbb{P}$. Similarly one can argue for $msg_2^{\beta}(x, C)_x \sim msg_2^{\beta_1}(x, C)_x$, $\forall x \in \mathbb{P}$.

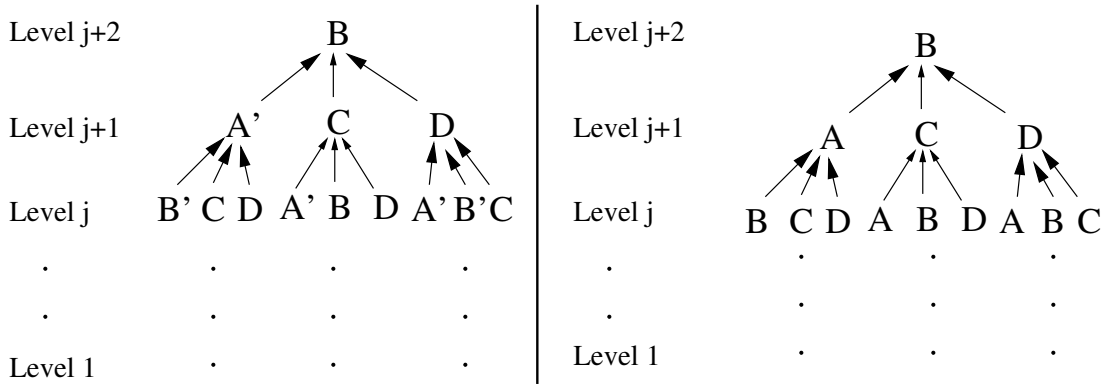

Figure 4.6: T_{β}^{B} and $T_{\beta_1}^{B}$ at the end of $j+1$ rounds.

Let the similarity be true till some round j i.e. $msg_i^{\beta}(x, B)_x \sim msg_i^{\beta_1}(x, B)_x$ and $msg_i^{\beta}(x, C)_x \sim msg_i^{\beta_1}(x, C)_x$, $\forall i | 1 \leq i \leq j$, $\forall x \in \mathbb{P}$. We now show that $\mathcal{A}$ can ensure that the similarity holds for round $j+1$ also. Consider T_{α}^{B} and $T_{\alpha_1}^{B}$ at the end of $k+1$ rounds as shown in Figure 4.6. To prove induction we need to show that B at level $j+2$ receives same messages in both trees. Consider node D at level $j+1$. From induction hypothesis C receive same messages till round j in both trees. Also 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 j i.e. $msg_j^{\beta}(C, D)_C \sim msg_j^{\beta_1}(C, D)_C$. For time being assume A' receives messages till round j in β_1 same as what A receives till round j 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. $msg_j^{\beta}(A', D)_{A'} \sim msg_j^{\beta_1}(A, D)_A$. Similarly assume that B' receives messages till round j in β_1 same as what B receives messages till round j 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 has insecure signature scheme in β_1, $\mathcal{A}$ can ensure that $msg_j^{\beta}(B', D)_{B'} \sim msg_j^{\beta_1}(B, D)_B$. Thus D at level $j+1$ receives same messages in T_{α}^{B} and $T_{\alpha_1}^{B}$. Since D is faulty in β_1, $\mathcal{A}$ can ensure that $msg_{j+1}^{\beta}(D, B)_D \sim msg_{j+1}^{\beta_1}(D, B)_D$. Using similar arguments one can show that $msg_{j+1}^{\beta}(C, B)_C \sim msg_{j+1}^{\beta_1}(C, B)_C$ and $msg_{j+1}^{\beta}(A', B)_{A'} \sim msg_{j+1}^{\beta_1}(A, B)_A$. Thus B receives same messages in round $j+1$ of β and β_1. Thus induction hypothesis holds for round $j+1$ too. Thus $msg_i^{\beta}(x, B)_x \sim msg_i^{\beta_1}(x, B)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$ holds true. Similarly one can argue for $msg_i^{\beta}(x, C)_x \sim msg_i^{\beta_1}(x, C)_x$, $\forall i > 0$, $\forall x \in \mathbb{P}$. The above proof is based on the assumption that A' upto level j 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' upto level j in T_{α}^{B} receives same messages as corresponding B in $T_{\beta_1}^{B}$. ∎

Lemma 35 *$view_B^{\beta} \sim view_B^{\beta_1}$ and $view_C^{\beta} \sim view_C^{\beta_1}$*

Proof: Follows from equation 4.3 and Lemma 34. ∎

We now give the adversary for β_2 and β_3 respectively. 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 upto 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).

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 $i-1$.)

Using aforementioned adversary and technique similar to one used in proof of Lemma 34, 35 one can prove the following four lemmas.

Lemma 36 *$msg_i^{\beta}(x,C)_x \sim msg_i^{\beta_2}(x,C)_x$, $msg_i^{\beta}(x,D)_x \sim msg_i^{\beta_2}(x,D)_x$ and $msg_i^{\beta}(x,A')_x \sim msg_i^{\beta_2}(x,A)_x$ $\forall i>0$, $\forall x \in P$.*

Lemma 37 *$view_C^{\beta} \sim view_C^{\beta_2}$, $view_D^{\beta} \sim view_D^{\beta_2}$ and $view_{A'}^{\beta} \sim view_A^{\beta_2}$*

Lemma 38 *$msg_i^{\beta}(x,A')_x \sim msg_i^{\beta_3}(x,A)_x$, $msg_i^{\beta}(x,B')_x \sim msg_i^{\beta_3}(x,B)_x$, and $msg_i^{\beta}(x,D)_x \sim msg_i^{\beta_3}(x,D)_x$, $\forall i>0$, $\forall x \in P$.*

Lemma 39 *$view_{A'}^{\beta} \sim view_A^{\beta_3}$, $view_{B'}^{\beta} \sim view_B^{\beta_3}$, $view_D^{\beta} \sim view_D^{\beta_3}$.*

Lemma 40 *There does not exists any BA protocol over a complete graph(G) of four nodes tolerating fault-structure $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$.*

Proof: Let there exists a BA protocol ϖ over a complete graph of four nodes tolerating fault-structure $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. We construct a system S(as described above) where each player runs ϖ' [Definition 12]. In scenario β_1, as per definition of a BA protocol [Definition 10], both B,C must eventually output value 0. From Lemma 35, B,C get same view in β and β_1. That is, for B,C β_1 is indistinguishable from β i.e. $\beta \overset{B}{\sim} \beta_1$ and $\beta \overset{C}{\sim} \beta_1$. Thus, B,C in β will eventually decide on value 0 (We are able to make claims regarding player's outputs in β as views of players are same in β and β_1. Thus by analyzing player's outputs in β_1, we can determine their outputs in β). Similarly using Lemma 39 A',B',D cannot distinguish between β and β_3 i.e. $\beta \overset{A'}{\sim} \beta_3$, $\beta \overset{B'}{\sim} \beta_3$ and $\beta \overset{D}{\sim} \beta_3$. Thus in β, A',B' and D eventually agree on value 1. Now consider execution β_2. As per agreement condition [Definition 10] A,C and D should decide on same value. From Lemma 37, $\beta \overset{A'}{\sim} \beta_2$, $\beta \overset{C}{\sim} \beta_2$ and $\beta \overset{D}{\sim} \beta_2$. Thus A',C and D should output same value in β as in β_2. However in β, C has already decided on 0 and D,A' have already decided on 1. This leads to a contradiction in S. From Lemma 33, it follows that our assumption that there exists a BA protocol ϖ over a complete graph of four nodes tolerating fault-structure $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$ is wrong. ∎

We now give the characterization of BA over complete graphs.

Theorem 41 *(t,k)-BA protocol over a complete graph G of n nodes exists if and only if $n > 2t + min(t,k)$.*

Proof: *Necessity:* We assume there exists a (t,k)-BA protocol η over a complete graph n nodes if $n \leq 2t + min(t,k)$. We show how to transform η into a BA protocol η' over a complete graph of four player $\{A,B,C,D\}$, 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 $min(t,k)$,

$min(t,k)$, t, $(t-min(t,k))$. adversary $\mathcal{A}$ can corrupt any of the following sets I_A, I_B, I_C, I_D, $(I_A \cup I_D), (I_B \cup I_D)$. Further, players in any of set I_A, I_B, I_D have insecure signature scheme broken. 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 show that η' is a BA protocol over a complete graph of four player $\{A, B, C, D\}$, tolerating $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. For simplicity we assign any corrupt players of η to be exactly those that are simulated by corrupted player in η'. Similarly, we assign players with insecure signature scheme in η to be exactly those that are simulated by players with insecure signature scheme in η'. Let ψ, ψ' be executions of η, η' respectively. We now show that if ψ satisfies specifications of a BA protocol [Definition 10], then so does ψ'. Let i,j $(i \neq j)$ be two non-faulty players in ψ'. i and j simulates at least one player each in ψ. *w.l.o.g* let them simulate players in I_i, I_j. Since i and j are non-faulty, so are all players in I_i, I_j. For ψ, all players in I_i, I_j must terminate, then so should i and j. In ψ, all non-faulty players including I_i, I_j should agree on same value say u, then in ψ', i, j also agree on u. In ψ' if all non-faulty start with value v, then so will all the non-faulty in ψ. In such a case in ψ, all non-faulty players including I_i, I_j should have $u = v$, then in ψ', i, j should have $u = v$. Thus ψ' also satisfies termination, validity and agreement conditions of a BA protocol [Definition 10]. Then η' is a BA protocol over four nodes tolerating fault-structure $\mathbb{A} = \{((A,D),(B)),((B),(A)),((C),(B))\}$. But from Theorem 40, we know that there cannot not exist any BA protocol tolerating $\mathbb{A}$ $= \{((A,D),(B)),((B),(A)),((C),(B))\}$. This contradicts our assumption that there exists a (t,k)-BA protocol η over a completely connected graph of n nodes when $n \leq 2t+min(t,k)$.

Sufficieny: Protocols given over incomplete networks work for complete networks as well. ■

Chapter 5

Conclusion and Future Work

We presented a complete characterization of Byzantine Agreement. When compared with the characterization of broadcast, it turns out, for the first time in literature, that there are graphs over which agreement is possible even though not all non-faulty players can reliably communicate with each other. In essence, *all-node global consistency is strictly easier than all-pairs point-to-point communication.* In this perspective, it appears that the problem of *agreement* could be a more fundamental primitive to general distributed computing than what (even the ubiquitous problem of) reliable communication is. The sufficiency proofs to our theorems contain the protocols for global consistency in various cases. The protocols are likely to be sub-optimal and there is a definite scope for improving the same. We, however, conjecture an exponential lower bound on the complexities of deterministic protocols.

Throughout the thesis, we work with *deterministic* protocols by which we mean that the protocol must have a zero-error probability in any run where (sufficient number of) signatures are unforgeable and correct. Since digital signatures themselves are typically probabilistic algorithms, it also makes sense to study BA protocols that have a small probability of error even in runs where the signatures do not fail. Hence, the introduction of randomness will be a interesting add on. Especially, given the fact that so far in the literature of Byzantine Agreement, randomization has had "tremendous influence/effect" on its (round as well as communication) complexities of Byzantine Agreement protocols. Along, the same lines we conjecture that the characterization would will more or less remain unchanged. However, we feel that the complexity of the protocols that use randomness will be significantly better than deterministic ones. We leave giving protocols with probabilistic correctness as an interesting open problem.

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. In *Computer science: research and applications*, pages 313–321, New York, NY, USA, 1992. Plenum Press.

[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, London, UK, 1990. Springer-Verlag.

[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.

[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.

[Can01] 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`.

[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.

[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, Massachusetts Institute of Technology, Cambridge, MA, USA, 1987.

[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.

[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.

[DDWY93] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly Secure Message Transmission. *Journal of the Association for Computing Machinery (JACM)*, 40(1):17–47, January 1993.

[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.

[Dol82] D. Dolev. The Byzantine Generals Strike Again. *Journal of Algorithms*, 3(1):14–30, March 1982.

[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.

[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] M. Fitzi and U. Maurer. Global broadcast by broadcasts among subsets of players. pages 267–, 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.

[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.

[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.

[HM97] M. Hirt and U. Maurer. Complete Characterization of Adversaries Tolerable in Secure Multi-party Computation. In *Proceedings of the 16th Symposium on Principles of Distributed Computing (PODC)*, pages 25–34. ACM Press, August 1997.

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

[KGSR02] M.V.N.A. Kumar, P. R. Goundan, K. Srinathan, and C. Pandu Rangan. On perfectly secure communication over arbitrary networks. In *Proceedings of the 21st Symposium on Principles of Distributed Computing (PODC)*, pages 193–202, Monterey, California, USA, July 2002. ACM Press.

[KK07] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. Technical report, University of Maryland, 2007.

[KY84] A. Karlin and A.C. Yao. Unpublished 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.

[Lin03] Y. Lindell. *Composition of Secure Multi-Party Protocols: A Comprehensive Study*, volume 2815 of *Lecture Notes in Computer Science (LNCS)*. Springer–Verlag, 2003.

[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.

[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.

[PCSR06] Arpita Patra, Ashish Choudhary, K. Srinathan, and C. Pandu Rangan. Constant phase bit optimal protocols for perfectly reliable and secure message transmission. In *INDOCRYPT*, pages 221–235, 2006.

[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.

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

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *FOCS '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.

[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.

[SAA95] H.M. Sayeed and H. Abu-Amara. Perfectly secure message transmission in asynchronous networks. *IEEE Symposium on Parallel and Distributed Processing*, pages 100–105, 1995.

[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.

[Sri06] Kannan Srinathan. *Secure Distributed Communication*. PhD thesis, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai, India, 2006.

[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.