UNIVERSIDAD DE GRANADA

Facultad de ciencias
E.T.S Ingeniería Informática y de Telecomunicación

Doble Bachelor's Degree in Computer Science and Mathematics

FINAL PROJECT

A Formal Model to Bitcoin Foundation

Student:
Baltasar del Sol de Haro

Tutor:
Juan Antonio Holgado Terriza
Departamento de Lenguajes y Sistemas Informáticos

Year 2022

To God, my family and friends.

Abstract

The present work strives to present a complete description of a protocol(the Bitcoin protocol) that uses the blockchain technology under the conditions of a formal computational model and analyze its properties in order to solve some two problems, namely the Byzantine Generals problem and the Public Transaction Ledger problem.

Sumario

El presente trabajo tiene como objetivo presentar una description completa de un protocolo(el protocolo Bitcoin) que utiliza la tecnología blockchain en el marco de un modelo computacional formal y el análisis de las propiedades del mismo para resolver dos problemas, el problema de los Generales Bizantinos y el problema de Mantener un Registro de Transacciones Público Distribuido.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

Technology has made a deep impact into everybody's life. Practically, everyone has the need to use a computer for work and lives surrounded by tens if not hundreds of processors with which they interact in one way or another, sometimes not even consciously. These computers are often times connected, creating a computing paradigm never seen before.

In this context of dependence and massive usage, the main service providers have become powerful players that users need to trust in order to use their sometimes essential services. One such example are banks. Banks have always been central players in the monetary territory that traditionally have been trusted with the responsibility of controlling inflation rate and loans. Their continuous failures have led to think if a less intrusive decentralized system is possible.

Blockchain is a new concept first used in the Bitcoin network described by Satoshi Nakamoto in his paper [Nak08]. Nakamoto buried his innovation within his implementation, known to the world as Bitcoin. This caused the resulting excitement to link the theoretical concept with the Bitcoin implementation. However, many definitions of blockchain in widespread are just incorrect. Often times because there is a confusion between specific use cases and implementation with the technology itself. Saeed Elnaj defines Blockchain as follows:

> "Blockchain is a digitized, distributed and secure ledger that guarantees immutable transactions and solves the trust problem when two parties exchange value."[Eln18]

This definition is misleading since it fuses cryptocurrencies and blockchain.

Valentina Gatteschi provides the following definition:

> "A Blockchain is a public ledger distributed over a network, recording transactions (messages sent from one network node to another) executed among networkparticipants. Before insertion, each transaction is verified by network nodes according to a majority consensus mechanism. Recorded information cannot be changed/erased and, at whatever time, the history of each transaction can be recreated."[Gat+18]

First of all, in both cases the blockchain described is the Nakamoto blockchain which embebes data into the blocks contrary to other blockchain like *Keyless Signature Infrastructure*

which do not do this. Second, the term distributed is used differently. Saeed Elnaj is describing a network architecture, while Valentina Gatteschi is stating that the Blockchain is distributed across the network independently of the network architecture. A generic Blockchain doesn't need to be distributed over a network. It is true that the distribution helps the computational security of the system but it is not mandatory.

Saeed Elnaj focuses on the security of the Blockchain. However, Blockchain is a concept, not a security protocol. A Blockchain is only as secure as the technical framework, the supporting infrastructure, and the length of the Blockchain. As a concept there are multiple ways to implement a Blockchain. Some authors as Don and Alex Tapscott describe the security features of the Blockchain as follows:

> "Safety measures are embedded in the network with no single point of failure, and they provide not only confidentiality, but also authenticity and nonrepudiation to all activity."[Rad18]

This makes little to no sense since the mentioned security properties are not inherent to the Blockchain, nor to the Bitcoin network. In any case they are *desirable* properties, but that is not what is stated. Also, confidence is talked about. But the confidentiality that is talked about is not the confidentiality of the transactions but the confidentiality of the persons conducting the transaction. This is important since there is historic precedence in which this confidence has been broken. Griffin and Shams describe two of the major heists:

> "Mt. Gox, a leading exchange that by 2013 was handling approximately 70% of bitcoin volume, declared bankruptcy due to a mysterious 'hack' of the exchange which resulted in approximately $450 million worth of bitcoin missing from investors' accounts. Good reasons have been put forward as to why the 'hack' may have been an inside job. ...In the second biggest hack in Bitcoin history, on August 2, 2016, the Bitfinex exchange announced that $72 million had been stolen from investor accounts, leading Bitcoin to plummet 20% in value."[GS20]

Felix Salmon talks about the first Bitcoin heist in 2011:

> "A man – we know him only as "All In Vain" – went to bed that night with his Windows computer turned on and connected to the internet. On that computer was a wallet containing 25,000 electronic coins. When he woke up on Monday morning, the wallet was still there. But the money was gone."

The "All In Vain" heist demonstrate security properties are not inherent to Blockchain applications and, in fact security problem arise from the fact that Blockchain are usually distributed across a P2P network which can spread worms in a matter of seconds. No technology is inherently secure, Blockchain is no exception.

Blockain is also stated to be inherently *immutable* and is often seen as the solution to every problem that requires such property. Truth is changing history is expensive, sometimes very expensive but never impossible. Again, history backs this fact when the Distributed Autonomous Organization was hacked as Nicholas Weaver tells and the Ethereum development team decided to reverse the heist by creating a fork in the Blockchain to undo the heist. Most Ethereum users approved this decision but not all of them, still, history was rewritten:

> "The first big smart contract, the DAO or Decentralized Autonomous Organization, sought to create a democratic mutual fund where investors could invest their Ethereum [Ether] and then vote on possible investments. Approximately 10% of all Ethereum ended up in the DAO before someone discovered a reentrancy bug that enabled the attacker to effectively steal all the Ethereum [Ether]. The only reason this bug and theft did not result in global losses is that Ethereum developers released a new version of the system that effectively undid the theft by altering the supposedly immutable Blockchain."[Wea18]

The real problem is well stated by Trevor I Kiviat who describes the problem in his writing for the Duke Law Journal:

> "Authors almost exclusively focus on bitcoin as a currency system. For example, authors have weighed the costs and benefits of transacting with virtual currencies, considered the sustainability of virtual currencies, and contemplated the application of existing regulatory schemes to virtual currency. Missing from the dialogue is a deeper perspective on the technology."

Although there is a lot of confusion around the Blockchain concept there is no doubt that when used to solve the appropriate problem it is a powerful and disruptive tool. Nevertheless it is necessary to formally define a theoretical model upon which a comprehensive protocol can be fully described and its properties perfectly analyzed. Proceeding this way it will be possible to archive Blockchain's full potential.

1.2 Objetives

Now the objectives of the work will be described. An analysis on the achievement of such objectives will be made in Chapter 7.

1.2.1 Primary Objective

The present work strives to present a complete description of a protocol(the Bitcoin protocol) that uses the Blockchain technology under the conditions of a formal computational model and analyze its properties in order to solve some two problems, namely the Byzantine Generals problem and the Public Transaction Ledger problem.

1.2.2 Secondary Objectives

1. Studying current state of the art works
2. Analyzing the most relevant current works on Blockchain topic from different standpoints
3. Exploring the accepted definitions of the Blockchain concept and the consequences of such definitions in terms of models and protocols
4. Searching for problems that can be solved using Blockchain properties
5. Defining in detail the computational model upon the protocol will be based

6. Describing the protocol that will use the Blockchain technology and will be used to solve the stated problems
7. Analyzing the vulnerabilities of the described protocol
8. Detailing the problems that are going to be solved
9. Illustrating the solutions given to the problems
10. Implementing the protocol in a general purpose language

1.3 Structure

The project structure is as follows. First a few concepts about distributed systems are presented lightly in Chapter 2. Then a formal description of the computation model the protocol will lie on is made in Chapter 3. In chapter 4 a concrete instantiation of the model is described and the complete protocol and its parameters follow. Then, in Chapter 5 some hypothesis are described as well as the desired properties which will be proved and analyzed in detail. In Chapter 6 some problems are described and the proven properties of the protocol are used in order to provide solutions for such problems. Last, in Chapter 7, a meditation on objectives completion is provided.

Chapter 2

Previous Concepts

In distributed systems there is often the need to take local decisions that satisfy global constrains. Such coordinated decision is called *agreement*. If the system is assumed to be perfect(i.e faultless) then reaching agreement is not only posible but often easy to achieve. Under the presence of faults though, the situation changes drastically. In this chapters the communication model will be lightly approach and such problems will be described and cataloged in a similar manner. This chapter is mainly based on [San06].

2.1 Network Synchrony

Network synchrony is related to the degree of coordination between the different components of the network. It is necessary to fix a level of synchrony before protocol development or analysis. There are three network synchrony conditions:

- **Synchronous**: Component operations are strongly coordinated. A synchronous network is modeled in rounds. This is often achieved using a centralized clock synchronization system. In any given round all components of the network perform exactly the same operations. For example, in round r all nodes may perform a measurement using a sensor and broadcast it in round $r + 1$.

- **Asynchronous**: Component operations are not coordinated in any way or form. This means component operations are not limited by coordination rules and are executed in an opportunistic manner. As a consequence there is no warranty of message reception since after no reply components have no way to know if it was due to a failure in the message transmission related to the network or the other component was simply busy.

- **Partially synchronous**: Component operations are not coordinated but there is an upper bound to message transmission delay. As a consequence message delivery is warranted but not in a timely way. Many protocols asume this level of synchronization since it is the most appropriate for networks based on an internet connection.

2.2 Faults and Failures

No system is perfect. Failures can occur in any system and studying, classifying and modeling such failures is necessary in order to improve the aforementioned systems. A failure or a fault is any behavior that deviates from the expected correct behavior. In distributed systems a failure and its cause may have very different nature. A failure can be caused by a flaw in the system's design, a programming error, a hardware error, an unexpected input... If a protocol is able to keep working in the presence of faults it is said to be *fault-tolerant* which makes it necessarily be more robust and reliable, both desirable properties. Designing *fault-tolerant* protocols can be a hard task due to the unpredictability and diverse nature of the failures. Also, protocol design increases in complexity as the components involved in the system grow. This means that designing protocols for big size networks like the internet is even harder.

2.2.1 Properties

Failures can be grouped in three different categories attending to their cause:

- **Execution**: Failures that occur in the execution of the protocol by a system component. Examples: computational errors, execution of the incorrect protocol step.
- **Transmission**: Failures that occur in the transmission system. Examples: loss and corruption of messages, message delivery to the wrong component.
- **Component**: Failures strictly related to a component and not the execution of the specific protocol. Examples: a link going down, a component shutting down.

The same failure can have different causes, thus can be classified differently according to the previous arrangement. For example; node n attempts to send a message to node m. If the message never arrives it could be due to an execution failure like node n failing to execute `send` operation or any other operation that precedes it in the protocol. It also could be caused by a transmission error like message loss by the transmission system. Finally the fault could be component related like the link between both components being down.

According to their time span failures can be classified in two groups:

- **Transient**: A fault that comes and goes out of existence. Transient failures may or may not reoccur. A transient fault that continues to reoccur is called intermittent. A corp that crosses through the beam of a wave transmitter causing a momentary disconnection is an example of a transient failure and a connector's loose contact is an example of an intermittent failure.
- **Permanent**: A fault that continues to exist until the underlying cause is repaired. Some examples are broken component pieces and software errors.

Attending to geographical spread failures can be classified in two groups:

- **Localized**: A fault only shown by a subset of the system components(a priori unknown).
- **Ubiquitous**: A fault that may be shown by any of the system components.

2.2.2 Failure Models

Let system properties and fault types be fixed, *resiliency* is defined as the maximum number of failures that can be tolerated by the system. In order to describe different types of failures some models need to be defined. Such models are defined attending to the previously stated failure cause classification.

Obviously, no protocol is resilient to any amount of failures. For instance, if all system's nodes stop working there is no protocol can be correct. Thus, the goal is to design systems that are able to withstand up to a certain amount of failures of a given type.

It is also important to note that the danger associated with a fault isn't dependant on its severity but rather on its impact on the system's behaviour and its detectability. As an example, a sudden broken server is a severe failure but it doesn't necessarily need to have a big impact on the system since a system with a synchronized server cluster can continue working with a slight impact on performance. On the other hand a tricky software error causing a bank's payment system to make deposits on the wrong account can have devastating effects since it is very hard to detect by nature and has a devastating effect on the system.

Component Failure Models

In this model the cause of a failure is a component and only components can fail. This model is the most commonly used in classic literature to discuss fault tolerance. Depending on which components fail three categories can be distinguished:

- **Entity Failures**: In this model only nodes can be faulty. This doesn't mean some faults not related to nodes can't be described using this model. For example, a link going down can be modelled as one of the nodes failing to execute operations `send` and `receive` with regards to the other node.
- **Link Failures**: In this model only links can be faulty. Similarly to *entity failures*, having only links fail doesn't mean some failures not related to links can't be described. For example, a node crash can be modelled as all its links crashing and a node computing wrong information and sending it to another node can be described as the link corrupting the message.
- **Hybrid Failures**: In this model both nodes and links can be faulty.

The least used model is the *hybrid model* since it is usually too complex due to the amount of failures that can be considered while the most used model is the *entity failures* model. In general, *entity failures* can be divided in three categories:

- **Crash**: A faulty node stops working completely, meaning it halts all execution. When a component fails it doesn't perform any other operation and stays off forever. It is important to emphasize that in this model a faulty component doesn't execute any wrong operation like sending corrupted or misleading messages.
- **Omission**: A faulty node can miss some of the received messages and/or doesn't send some of the prepared messages. In the first case the fault is known as send omission

and the second case is known as receive omission. These type of faults are usually caused by memory overflow.

- **Byzantine Failure**: A faulty node can exhibit any behavior. Software bugs often produce this kind of failures. Also, a faulty node may actually be *malicious* executing operations in an attempt to damage system's behavior and can even be coordinated with other faulty components to cause a greater impact.

A few observations are worth doing; First, note that *crash* failures are a type of *omission* failure, particularly a permanent send/receive *omission* failure. Second, *byzantine* failures are an extreme kind of failure that includes *omission* failures. Thus, there exists an inclusion order between the three types of failures with *byzantine* failures being the biggest set of the three. As a consequence, a system resilient to *byzantine* failures can tolerate any other failure.

Communication Failure Models In this model the cause of a failure is the communication subsystem. The communication subsystem can produce the following failures: message loss, message corruption, message delivery to the wrong component. In the *communication failure* model the communication subsystem is bounded to only three types of faults:

- **Omission**: A sent message is never delivered.
- **Addition**: A message is received although none was sent.
- **Corruption**: The received message doesn't coincide with the one sent.

The use of *omission* and *addition* faults in the description of several failures is quiet obvious, not so much the use of the *addition* fault. Several failures need to use *addition* error to be correctly described. For example, noise in the communication channel could be confused with a message. Even more important is its use in modelling "non-authorised" messages which are messages sent by a not-authorised user. Spam is a clear example of this phenomenon.

There is no clear hierarchy regarding the danger associated with these failures. The hierarchy comes naturally when two or all of these faults can occur in the system at the same time. When all faults occur at the same time such behavior is called a *Byzantine* failure.

Chapter 3

Computation Model

In the current chapter a formal model will be described that can be classified using the concepts explained in Chapter 2. This model is a personal take on [Can01] which is the model used in the article this work is mainly based on [GKL15].

3.1 Model Definition

A protocol is a computer program, intended to be executed by a set of computational entities or *parties* with the ability to communicate between them. A protocol execution is a set of interacting computing elements, each running the protocol on its own input and taking its own random choices. The *adversary* noted with $\mathcal{A}$, is a computational entity that controls a subset of the parties and has some kind of control over the communication network. It represents a malicious entity trying to prevent parties from behaving correctly in terms of some protocol properties or the protocol objective itself. The *environment*, noted by $\mathcal{Z}$, represents any external external condition to the current protocol like other protocol and their adversaries, human users, etc. The environment is allowed to interact with the adversary and the rest of the parties at any point of the execution. All parties interact until each one generates a local output. The concatenation of the local outputs of the adversary and all parties is called the *global output*.

3.1.1 Interactive Turing Machines

In order to model the computational objects that parties represent an extension of the traditional Turing machines will be used. *Interactive Turing Machines* are an extension on the standard Turing machine created to capture the casuistry of a distributed algorithm, namely a protocol.

Definition 1. An *interactive Turing machine(ITM)* μ is a Turing machine with the following augmentations:

New Data Structures:

- `Identity tape`: This tape is "read only". Its content is interpreted as two strings. The first string is a description of the program run in the on μ, i.e, its state transition

function and initial tape contents. This is called μ code. The second string is an identifier called *identity* of μ. The code and the identity are called the *extended identity* of μ.

- `Outgoing message tape`: This tape holds the current outgoing message along with enough addressing information for message delivery.

- `Input tape`: This tape is "read only". This tape represents incoming messages.

New Instructions:

- `External write`: The effect of this instruction is that the message currently written on the outgoing message tape is possibly written to the input tape of the machine with the identity specified in the outgoing message tape.

- `Read next message`: The effect of this instruction is that the reading head jumps to the beginning of the next message on that tape. In order to implement this instruction it can be supposed that each instruction ends with a special character.

Definition 2. A *configuration* of an ITM μ consists of the contents of all tapes, as well as the current state and the location of the head in each tape. A configuration is active if the activation tape is set to 1, else it is inactive.

Each party on the system will be modeled as ITM, including the ones controlled by the adversary $\mathcal{A}$. In the *bitcoin backbone protocol* $\mathcal{A}$ will have some communication advantages over the honest of the parties. In order to model this situation a control program, noted C, has to be introduced. $C : \{0,1\}* \rightarrow \{allow, deny\}$ takes any configuration of an ITM and an ITM identity and returns a value meaning the function has or not permission to directly communicate with ITM. Now, communication between parties can be formally defined.

Communication Communication between parties is based on the `External write`. Let $M = (\mu, id)$ denote the ITM which executes instruction `External write`. Then the current contents of the outgoing message is interpreted as a sequence of tuples:

$$(id, id', r, m)$$

Where id is the machine's id, id' is an ITM identity, r is the `reveal-sender-id` flag and $m \in \{0,1\}*$ is the `message`. Iteratively over the sequence of tuples, consider the output of function C when given (id, id', r, m) as input. If this is $disallow$, then the instruction is not carried out and the calling machine is halted. If C outputs $allow$, then if an ITM M' with identifier id' exists on the system then message m is written to the input tape of M'. If the `reveal-sender-id` flag is set, then the identity of M is also written to the input tape of M'. The control program C may output $disallow$ in several cases like the id written on the message not coinciding with the real machine address(which can't be changed since the `identity tape` is read only) or if the sender machine does not have permission to input a communicate with the addressee machine.

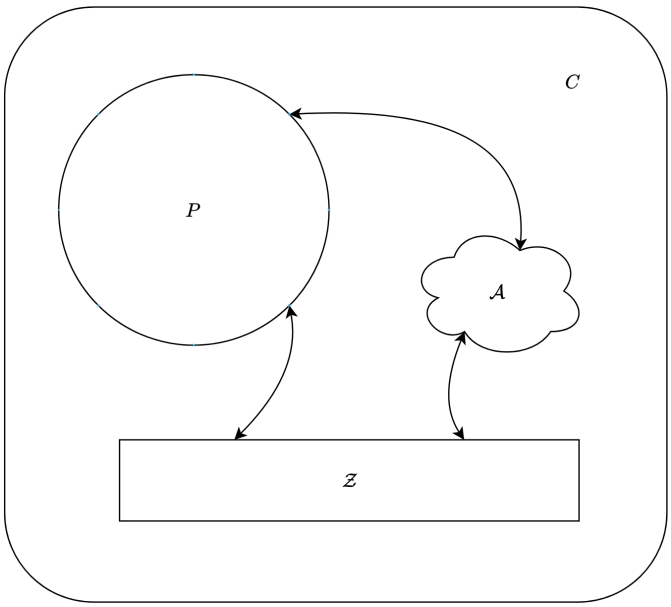

Figure 3.1: Model sketch

Polynomial Time ITMs A Turing machine μ is said to be T-bounded if, given any input of length n, μ halts within at most $T(n)$ steps. This guarantees that an execution of a system of ITMs completes in bounded time. An interactive Turing machine μ is said to be T-bounded if, given any input of length n, μ halts within at most $T(n) + T(m) - T(m')$ steps where m is the number of received messages during its execution and m' is the number of sent messages.

3.1.2 Services

During the definition of the *Bitcoin backbone protocol* there will be a need to use certain certain processes that are are different in their purpose and scope from the protocol parties. A *function* will be the equivalent of a mathematical function, it maps inputs to outputs. A *functionality* is different from a function in the sense that it maintains a `state` that may be updated with each call. Last an *oracle* is a functionality that has a random component to it. All of these entities will be modeled by ITMs each with a special identifier that allows parties in the system to identify and communicate with them. Since they are modeled by ITMs they will use function `Read next message` to read the next message in its input and tape, process it and write the pertinent output to the senders ITM which must have specified its id and set $r = 1$ in its message. By this mechanism it can be said that these services can be accessed in a concurrent manner. As stated these processes are not considered parties.

3.1.3 Protocol execution

A *system of ITMs* is a pair $S = \langle Z, C \rangle$ where Z is an environment and C is a control function. Parties and services have their code written in their identity tape but not their identity. The

execution of a system $S = \langle Z, C \rangle$ goes as follows. First, the control program writes a unique identity for each party in Z and each service. Next C sends the start signal to all parties. Parties execute their code and interchange messages following the rules coded in C until they obtain an output, then they execute a special halt instruction which writes their output on C and halts the machine. When all parties have written their respective output on the input tape of C the initial machine writes a final message in Z and halts. Execution is ended.

Chapter 4

The Bitcoin Backbone Protocol

This chapter has three main sections. In the first one a concrete instantiation of the model described in Chapter 3 is given. In the second section a description of the Blockchain underlying data structure is provided. Last the protocol is properly defined and parameterized. This chapter is mainly based on [GKL15].

4.1 Model Instantiation

The following model instantiation is one chosen in order to model the conditions usually present in the settings the protocol will be executed. The decision taken try to model such conditions in the simplest, yet most concrete possible manner.

A set of parties P, an adversary $\mathcal{A}$, an environment $\mathcal{Z}$, a control program C and a protocol Π are considered. The total number of parties n is defined from the start as well as the number of parties control by the adversary t. Three services, a functionality and two random oracles, need to be defined in order to model network communication, hashing power and the capacity to create random strings called *nonces*:

- Diffuse functionality: In order to provide better readability without loss of generality the INPUT tape of each party will be split in two:
  - Communication tape. Noted as RECEIVE. It will be used to store messages coming from other parties.
  - Input tape. Noted as INPUT. It will be used to store messages coming from the environment $\mathcal{Z}$.

  The diffuse functionality maintains a *round* counter which is initialized at 1. The diffuse functionality acts like a buffer that stores parties messages and delivers them at the end of the communication stage: When the functionality receives an instruction to diffuse a message m from a non-adversary party P_i it marks the party as complete for the current round. When all parties are complete for the current round, the functionality writes all messages into the RECEIVE tape of each corresponding party. The *round* counter is incremented. The adversary is excluded from this buffer mechanism, it can write messages directly in the RECEIVE tape of each party and it can fetch

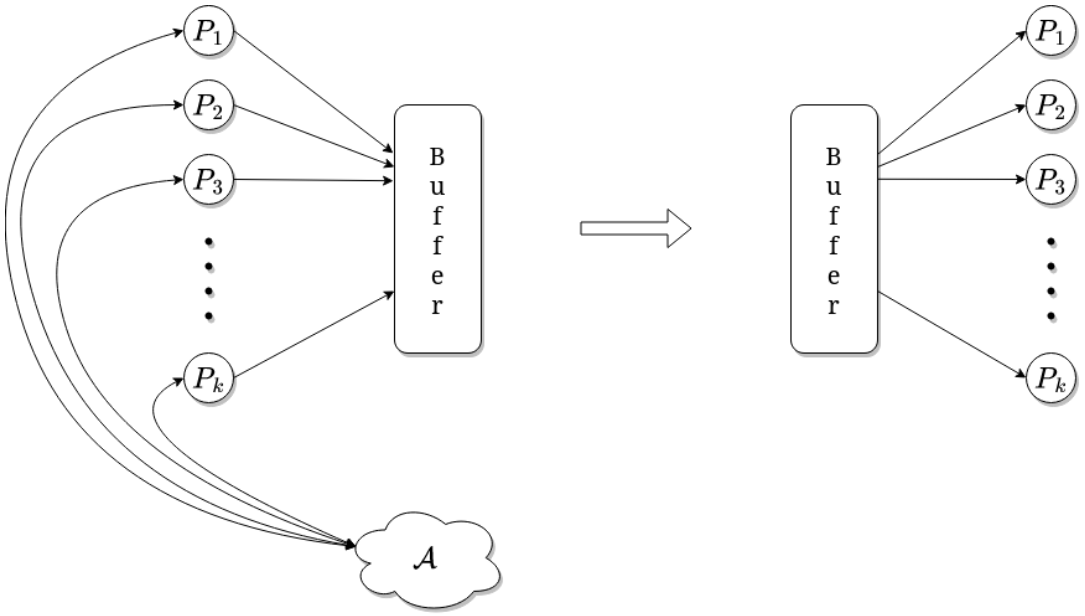

Figure 4.1: Diffuse functionality diagram

messages from the buffer at any point but has to send a complete message to the functionality for it to deliver honest parties messages. An illustration of the diffuse functionality can be seen in Figure 4.1.

- Nonce generator oracle($A(\cdot)$): The nonce generator oracles generates a *unique* random string in $\{0,1\}*$. The nonce is unique in the sense that no two calls to the nonce generator oracle return the same string. This property will be referred to as *nonce entropy*. The reason for nonce entropy is discussed in Remark 3.

- Hash function oracle($H(\cdot)$): This service has two modes:

  - `Calculation`: When queried in calculation mode it receives an input value x, if x has not been queried before a random value in $y \in \{0,1\}^*$ is returned. The oracle keeps a table in which pair (x, y) is stored. If the value x has already been queried its pair y stored in the table is returned.Every honest party P_i is given q calculation queries each round. The adversary $\mathcal{A}$ is given $t \cdot q$ queries each round. The environment $\mathcal{Z}$ is not given any queries.
  - `Verification`: When queried in verification mode the oracle expects two values x and y and the oracle returns `true` if pair (x, y) is stored on the table, otherwise `false` is returned.

The fact that parties cannot communicate directly with each other but have to make use of a diffuse functionality is an attempt to model Bitcoin't P2P network layer. The fact that the adversary can bypass the diffuse buffer by accessing any parties messages before committing theirs and writing their message directly in the RECEIVE tape is captured

by saying the adversary has "rushing"[1]capabilities. On the other hand, the adversary cannot change the content of the messages sent by honest parties or prevent them from being delivered. This mimics communication over TCP/IP in the Internet in the sense that messages between parties are reliably delivered but malicious parties may change the message sender that goes through them making it appear it originates from a different source.

A model may be considered where not all nodes can receive all messages reliably. This can be achieved using this same model and letting the adversary control these players and simulate them honestly while not performing the `read-next-message` instruction properly at random.

The following subsection justifies there is no need to impose an strict an upper bound on the number of messages. More on Remark 1.

4.1.1 q-bounded synchronous setting

$\{\texttt{EXEC}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}$ will be used to denote the random variable ensemble that determines the output of environment $\mathcal{Z}$ for a protocol Π that uses the services previously described: diffuse functionality, nonce generator oracle and hash generator oracle. $\{\texttt{EXEC}^{P_i,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}$ will be used to denote the random variable ensemble describing the view of party P_i after the completion of an execution with environment $\mathcal{Z}$, running protocol Π and adversary $\mathcal{A}$. Furthermore, $\texttt{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}} = \cup_{i=1,\ldots,n}\langle\texttt{VIEW}^{P_i,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}\rangle$.

Note that parties not need to be aware of the number of parties executing the protocol and, because of the unauthenticated nature of the network only protocols that do not make explicit use of the number of parties n or their identities. Also note that n and which parties are controlled by the adversary $\mathcal{A}$ is fixed during the execution of the protocol and they will be hard-coded in the control program C[2].

Parties limited ability to produce PoWs is modeled by the limit imposed to all parties in their access to the hash function. In this model all parties computational power is supposed to be equal since all parties are given q computing hash queries per round. In the real world different parties may have different computational power but this can be replicated by the model considering a parties computational power as a unit of power and having real world parties represented by clusters of one unit parties. Note that q is an upper limit, it doesn't have to be reached. The reason for not giving the environment $\mathcal{Z}$ any computational power is for the adversary's computational power to be proportional to the number of parties controlled by the adversary. Thus, the environment can still help the adversary and the invested resources can be distributed in case the same hashing function is being used by another protocol running in parallel but computational power is still tied to the number of parties controlled by the adversary.

[1]Although the rushing capability it is strong and has some important consequences that will be stated latter it is worth saying that it is actually argued to be realistic by properly distributing malicious nodes in the network whose only task is to echo adversary messages.

[2]In reality, the number of parties controlled by the adversary has to be fixed but not which ones as it is demonstrated in [GKL15].

4.1.2 Protocol Properties

Theorems will be related to *properties* of protocols in the q-bounded synchronous setting. Properties will be defined as predicates over the random variable ensemble $\texttt{EXEC}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ restricting it to adversaries $\mathcal{A}$ and environment $\mathcal{Z}$ that are polynomially bounded. Such predicates will verify with high probability on κ. The probability space is determined by the choices of the random oracles of all ITMs.

Definition 3. Given a predicate Q and fixed $q, t, n \in \mathbb{N}$ with $t < n$, *protocol* Π *satisfies property* Q *in the* q*-bounded setting for* n *parties assuming the number of corruptions is bounded by* t if, for all polynomial-time $\mathcal{A}, \mathcal{Z}$ the probability that $Q(\texttt{EXEC}^{P_i,t,n}_{\Pi,\mathcal{A},\mathcal{Z}})$ is false is negligible in κ.

Definition 4 (Negligible function)**.** A *negligible function* is a function $F : \mathbb{N} \to \mathbb{R}$ such that, for every positive integer c there exists an integer n_c such that $|\mu(x)| < \frac{1}{x^c},\ \forall x > n_c$.

Intuitively a function is negligible when it is *eventually* smaller than the inverse of c power function(could be extended to a polynomial of grade c) which means it decreases faster than the c power function grows. This notion is very common in cryptography and is strictly linked to the definition of continuous function, is allows to claim that error probability decreases "very" fast. To simplify the discussion z in $\{\texttt{EXEC}^{P,t,n}_{\Pi,\mathcal{A},\mathcal{Z}}(z)\}_{z\in\{0,1\}^*}$ will be fixed to 1^κ. Therefore, the whole security analysis depends on κ and it will be the known as the *security parameter*. Note that the number of parties controlled by the adversary t and the total number of parties in the system also have a huge impact on the security analysis but they are not a parameter since they can't be change by the protocol.

Remark 1. Because computational resources are bounded there is no need to impose an upper bound on messages since sending messages means spending computational resources, thus it provides no advantage and the adversary has no way to turn this in its favor.

4.2 Blockchain Definition

Let $H(\cdot)$ be a cryptographic hash function with output $\{0,1\}^\kappa$. A *block* is a triple $B = \langle \alpha, \rho, x \rangle$. A block will said to be valid if it that satisfies predicate $\texttt{validblock}^T(B)$ defined as:

$$H(\alpha, \rho, x) < T$$

Where:

- $\alpha \in \mathbb{N}$, represents a *nonce*. It will be used to generate different inputs for the hash function in order to obtain a combination that satisfies $\texttt{validblock}^T(B)$ once ρ and x are fixed.
- $\rho \in \{0,1\}^\kappa$, represents a hash reference to the previous block. It will be used in order to ensure the computational efforts needed to extend a block called PoW cannot be used at the same time to compute more blocks.
- $x \in \{0,1\}^*$, represents the *content* of the block.

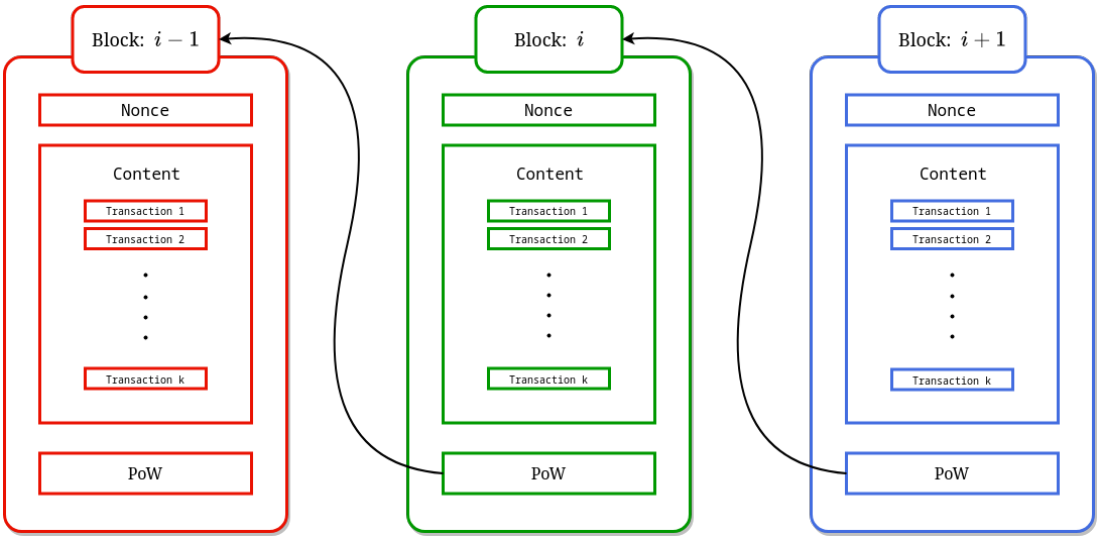

Figure 4.2: Blockchain sketch

- $\mathbb{N} \ni T < 2^{\kappa}$, represents the *difficulty level*. This value may change across different blocks in order to adjust the rate at which new blocks are produced. This is useful in order to maintain an steady mining rate in case the number of parties may change between rounds.

A *Blockchain* or a *chain* is a tuple of blocks $\mathcal{C} = \langle B_1, \ldots, B_n \rangle$. By convention the empty tuple ϵ is also a chain. The length of a chain C is its number of blocks and is noted by len(C). By convention len(ϵ)$= 0$. Given a chain C with len(C)$= n > 0$, its *content* is defined as a n-tupla $x_{\mathcal{C}} = \langle x_1, \ldots, x_n \rangle$ where x_i represents the x-values of block B_i in chain C. The last block of a chain C is called the *head* of the chain is noted head($\mathcal{C}$). By convention head(ϵ) $= \epsilon$, the empty block.

Given a chain $\mathcal{C} = \langle B_1, \ldots, B_m \rangle$ and $k \in \mathbb{N}$, let $\mathcal{C}^{\lceil k} = \langle B_1, \ldots, B_{m-k} \rangle$, i.e, the chain resulting from erasing the last k blocks from $\mathcal{C}$. If $k \geq m$ then $\mathcal{C}^{\lceil k} = \epsilon$ by convention. Also, given two chains $\mathcal{C}_1, \mathcal{C}_2$ it will be said $\mathcal{C}_1$ is a prefix of C_2 if exists $k \in \mathbb{N}$ such that $C_2^{\lceil k} = C_1$.

A chain $\mathcal{C}$ with head($\mathcal{C}$) $= \langle \hat{\alpha}, \hat{\rho}, \hat{x} \rangle$ may be extended by appending a block $B = \langle \alpha, \rho, x \rangle$ such that $\rho = H(\hat{\alpha}, \hat{\rho}, \hat{x})$. This condition is known as proof of work and abbreviated as PoW. Only a block with $\rho = 0$ can extend an empty chain ϵ[3]. A chain C extended by block B it is noted as $\mathcal{C}_{new} = CB$ and verifies head($\mathcal{C}_{new}$)$= B$.

Remark 2. ρ is a hash value, thus, given a block B there is no way to know which block precedes B it is also given as a chain in which case it can be checked the blocks are properly

[3]The first block of a chain is called the Genesis block. It doesn't point to any other block and it usually contains some trusted setup information hard-coded in the source of the application. In the original Bitcoin protocol the Genesis block, among other information, contains the following sentence: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" which is the main headline of The Times issue of such date. The quote has two purposes, in the first place, it demonstrate the block was mined after such date and, secondly, it points at the financial instability as a likely cause for the Bitcoin development.

"linked".

4.3 Protocol Definition

4.3.1 External Functions

The external functions are:

- Content validation predicate $V(\cdot)$: Given a chain $\mathcal{C}$ the content validation predicate receives $x_{\mathcal{C}}$ as input and returns 1 if the content is consistent with the application built on top of the protocol. Otherwise 0 is returned. By convention $V(\epsilon) = 1$.

- Input contribution functionality $I(\cdot)$: The input contribution function receives tuple $\langle \mathcal{C}, \text{INPUT}(), \text{RECEIVE}() \rangle$ as input, where C is a chain, INPUT() are the contents of the input tape and RECEIVE() the contents of the network tape. The function will produce some block content x . The purpose of x is to be used in the making of a block that extends $\mathcal{C}$, thus, it is sensible to ask for x to be consistent with the content of $\mathcal{C}$, i.e, for any chain $\mathcal{C}$ with $x_{\mathcal{C}} = \langle x_1, \ldots, x_n \rangle$ a value x produced by $I(\mathcal{C}, \cdot, \cdot)$ must satisfy $V(\langle x_1, \ldots, x_n, x \rangle) = 1$. It is important to note that this functionality can preserve state in the form of a table or any other structure that may be used in any other future invocation.

- Chain reading function $R(\cdot)$: The chain reading function receives a chain C as input and generates some kind of output dependent on the purpose of the application built on top of the protocol. In its simplest form this function returns the chain's contents. The importance of the existence of this function can be debated. In the protocols described in Chapter 6 this function is quiet simple and only uses information relative to the chain's contents but it doesn't have to be the case. In case information related to the chain other than its content is needed in order to create an output then the chain's content will not be enough and this function's existence is mandatory.

In Figure 4.3 the basic operations of the bitcoin backbone are illustrated. Party P_1 receives READ instruction from the environment $\mathcal{Z}$ that resolves returning the value output by function $R(\cdot)$ which is the sequence $\langle x_1, \ldots, x_5 \rangle$. Party P_2 receives instruction INSERT value val which resolves calling functionality $I(\cdot)$ in order to produce some valid content y_5 to extend its current chain. It calculates a new block successfully and, after appending it to its current chain, the resulting chain is broadcast. Party P_3 receives the chain calculated by P_2 and proceeds to check its content via the content validation predicate and its correct structure. If the chain is correct it will need to decide on which chain to keep. Last, note that the combined view of parties P_1, P_2, P_3 is inconsistent but they agree on a common prefix $\langle x_1, x_2, x_3 \rangle$.

4.3.2 Auxiliary Algorithms

Three algorithms will be necessary in order to build the protocol:

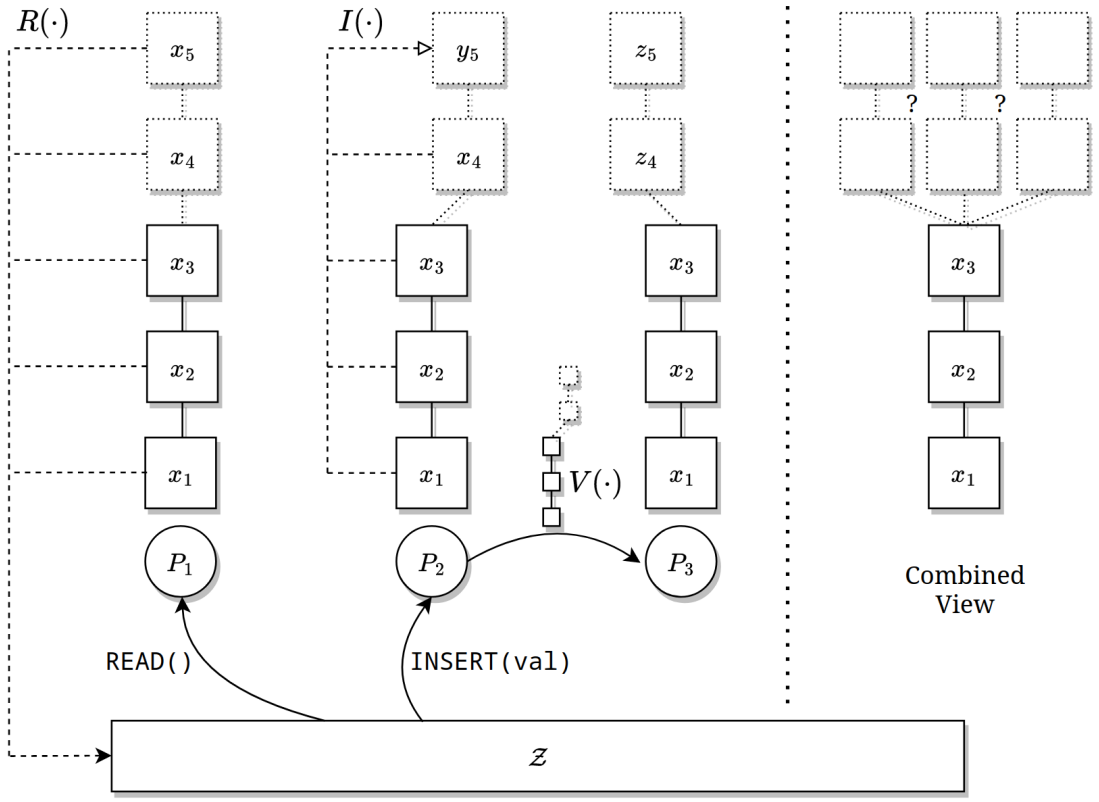

Figure 4.3: Overview of basic operations

Chain validation. The algorithm called `validate_chain` performs a validation of the structural properties of a chain. It takes a chain C, the value T and a hash function $H(\cdot)$ as input. It is parameterized by the validation predicate $V(\cdot)$. The algorithm verifies every blocks in C is valid by checking it fulfills the $validblock^T$ predicate and also verifies every block extends the previous one by certifying its PoW. If all blocks are verified and their content is consistent, i.e, $V(x_{\mathcal{C}}) = 1$, the chain is deemed as valid, and the algorithm returns 1. Otherwise the chain is rejected and the algorithm returns 0. Its pseudocode is represented in Algorithm 1.

Algorithm 1 *Chain validation predicate* algorithm. Parameters are T, the cryptographic hash function $H(\cdot)$ and the content validation predicate $V(\cdot)$. Input is $\mathcal{C}$.

```
1: function validate(C)
2:     if C = ε then
3:         return True
4:     end if
5:
6:     if V(x_C) then
7:         B ← head(C)
8:         ρ ← H(B)
9:         repeat
10:            C ← C^⌈1
11:            B' ← head(C)
12:            ρ' ← H(B')
13:            if (validblock^T(B) = False) ∨ (ρ ≠ ρ') then
14:                return False
15:            end if
16:            ρ ← ρ'
17:            B ← B'
18:        until (C = ε)
19:    end if
20:    return True
21: end function
```

Chain comparison. The second algorithm is called `maxvalid` it utilizes `validate_chain` and its purpose is to select the "best" chain out of a set of chains. The algorithm iterates on the set of chains comparing them by pairs using the function `max` which encodes some kind of order in the space of chains. The main factor used by function `max` in order to determine the best chain is the length. When two chains have the same length some tie breaking criteria needs to be applied. There are many options but the one chosen will be to always return the first chain. Since the set of chains is ordered by arrival time, this rule means any party will prefer its local chain opposed to any other chain. This is consistent with the current Bitcoin operation. Since solving PoW takes computational power the longest chain criteria means parties are always looking forward to obtain(and thus extend) the current longest chain which is the one that amasses the most computational power. This

characteristic will be key in the following analysis. The algorithm's pseudocode can be seen in Algorithm 2.

Algorithm 2 *Chain comparison* algorithm. Parameters is `max(·)`. Input is $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$.

```
1: function maxvalid({C_1, ..., C_k})
2:     aux ← ϵ
3:     for i=1: k do
4:         if validate(C_i) then aux ← max(C_i, aux)
5:         end if
6:     end for
7:     return aux
8: end function
```

Proof of Work. The purpose of the third algorithm referred as `pow` obtain a block such that verifies the PoW condition. The algorithm takes some content value x and a chain C as input. It is parameterized by the values q and T and a cryptographic hash function $H(\cdot)$ and the functionality $A(\cdot)$. The algorithm will use functionality $A(\cdot)$ in order to generate up to q random nonces to which it will append input x as well as the hash identifier of the current head of the chain to build a valid block which accomplishes PoW. If one of the q trials is successful the chain extended with the calculated block is returned; otherwise the current chain is returned. The algorithm is illustrated in Algorithm 3.

Algorithm 3 *Proof of work* algorithm. Parameters are q, T, a cryptographic hash function $H(\cdot)$ and the nonce generator functionality $N(\cdot)$. Input is $(\mathcal{C}, x)$.

```
1: function pow(C, x)
2:     if C = ϵ then
3:         ρ ← 0
4:     else
5:         ρ ← H(head(C))
6:     end if
7:
8:     for i=1: q do
9:         α ← A()
10:        if H(⟨α, ρ, x⟩) < T then
11:            B ← ⟨α, ρ, x⟩
12:            C ← CB
13:            break
14:        end if
15:    end for
16:    return C
17: end function
```

Remark 3. If two parties could obtain the same PoW then the adversary $\mathcal{A}$ could have and advantage since they could share their history of hashed values constantly and this could

give them the upper hand in calculating PoWs since honest nodes could spend their hash queries in exactly the same values. Obviously this is not very relevant and the probability of such event decreases exponentially in κ, still, the nonce entropy hypothesis is not too strong and it greatly simplifies the discussion.

Remark 4. The adversary cannot ask for help in the calculation of PoW to an external process since the state of the hash calculation oracle cannot be replicated in any way. Also note that this is not restrictive since there additional computational power can be modeled by increasing t value.

4.3.3 The Backbone Protocol

Now the Bitcoin backbone protocol will be described using the functions described previously. The protocol receives no input. The protocol is parameterized by two functions, the input contribution function $V(\cdot)$ and the chain reading function $R(\cdot)$. The first step is to initialize the variables. Then an "infinite" loop is executed although the following analysis is only valid in case the total polynomial time is polynomial in κ(as it has been already stated). The loop has two main sections.

In the first section, the best valid chain $\mathcal{C}$ is selected between its local chain and the ones available in the communication tape RECEIVE() using function `maxvalid`, the other chains are discarded. Some block content x is created invoking functionality $I(\cdot)$ over the current chain. Note that, as stated before, functionality $I(\cdot)$ may use any kind of knowledge about the state of the execution and the contents of the communication tape RECEIVE() and the input tape INPUT(). Then, the party executing the protocol attempts to extend its chain $\mathcal{C}$ with content x calling `pow` function. If $\mathcal{C}$ is successfully extended the resulting chain is broadcast using functionality DIFFUSE(), otherwise an end of round signal is issued(represented by a termination message $\perp$).

In the second section, if the input tape INPUT() contains value READ the output of function $R(\cdot)$ over the current chain $\mathcal{C}$ are written in the OUTPUT() tape. Finally round counter is increased.

Two clarifications are worth doing. Although the protocol is executed indefinitely the incoming analysis only applies when the total running time is polynomial in κ. In this particular version of the protocol only two types of entries expected in the input tape: (INSERT, $value$) and READ; other inputs are ignored.

Algorithm 4 *Bitcoin backbone* protocol. Parameters are the input contribution function $I(\cdot)$ and the chain reading function $R(\cdot)$. Input is none.

```
1:  function protocol
2:      C ← ε
3:      round ← 1
4:
5:      while True do
6:          C ← maxvalid(C, chains in RECEIVE())
7:          x ← I(C, INPUT(), RECEIVE())
8:          Ĉ ← pow(C, x)
9:          if C ≠ Ĉ then
10:             C ← Ĉ
11:             DIFFUSE(C)
12:         else
13:             DIFFUSE(⊥)
14:         end if
15:
16:         if INPUT() contains READ then
17:             write R(C) to OUTPUT()
18:         end if
19:
20:         round ← round + 1
21:     end while
22: end function
```

Chapter 5

Protocol Properties

This chapters has two main sections. In the first section all necessary definitions and preliminary results are developed. In the second section the main properties of the Blockchain protocol described in Chapter 4 are defined, illustrated and proved. This chapter is mainly based on [GKL15].

5.1 Preliminaries

In order to complete a PoW a party needs to find a value combination $H(\alpha, \rho, x)$ such that $H(\alpha, \rho, x) \leq T$. A query in which such value is found will be called *successful*.

Definition 5 (Random Variables)**.**
For every round $i \in \{1, \ldots, \lambda\}$, query $j \in \{1, \ldots, q\}$ and adversary $k \in \{1, \ldots, t\}$ the following random variables are defined; If at round i an honest party succeeds at solving POW, then $X_i = 1$, otherwise $X_i = 0$. If at round i exclusively an honest party succeeds at solving POW, then $Y_i = 1$, otherwise $Y_i = 0$. If at round i, at query j the adversary k obtains a successful query, then $Z_{ijk} = 1$, otherwise $Z_{ijk} = 0$. Also $Z_i = \sum_{k=1}^{t} \sum_{j=1}^{q} Z_{ijk}$. Consequently, for a set of rounds S, $X(S) = \sum_{s \in S} X_i$, similarly for $Y(S)$ and $Z(S)$.

A round such that $X_i = 1$ will be called a *successful round*, a round such that $Y_i = 1$ will be called a *uniquely successful round*. Now a set of events will be described. These events are extremely problematic since the possibility of their occurrence disrupts the plot of some proofs.

Definition 6 (Disruptive Events)**.**

- An *insertion* occurs when, given a chain which contains two consecutive blocks B_1 and B_2, a block B_3 is calculated such that B_1, B_3, B_2 constitute three consecutive blocks of a valid chain.
- A *copy* occurs when the same block B is present in two different positions.
- A *prediction* occurs when a block extends one which was computed at a later round.

In our model(as long as the executions is polynomially bounded in κ) these events are extremely unlikely(their probability is exponentially small in κ). A typical execution is one in which these events do not occur and the sum of the random variables X, Y, Z does not deviate too much from their expected value in any big enough subset of rounds.

Definition 7 (Typical Execution)**.** *An execution is (ϵ, λ)-typical, for $\epsilon \in (0,1)$ and $\lambda \in \mathbb{N}, \lambda \geq 2/f$, if, for any set S of at least λ consecutive rounds, the following hold:*

(a) $(1-\epsilon)\mathbb{E}[X(S)] < X(S) < (1+\epsilon)\mathbb{E}(S)$.

(b) $(1-\epsilon)\mathbb{E}[Y(S)] < Y(S)$.

(c) $Z(S) < \mathbb{E}[Z(S)] + \epsilon\mathbb{E}[X(S)]$.

(d) No disruptive events occurred.

Remark 5. *The fact that a typical execution is only defined for $\lambda \geq 2/f$ may look arbitrary now but will be an important hypothesis in order to prove Lemma 2.*

Theorem 1. *An execution is typical with probability $1 - e^{-\Omega(\epsilon^2\lambda f + \kappa - log(L))}$. Where L is the total runtime of the system(the total number of rounds).*

Remark 6. Ω notes a positive transformation and is used to stress the parameters of the probability of a typical execution and their relation while obscuring irrelevant constants. It is easy to see that $1 - e^{-\Omega(\epsilon^2\lambda f + \kappa - log(L))}$ is negligible in κ since functions $n \mapsto a^{-n}$ are negligible $\forall a \geq 2$.

In order to bound the random variables expectancy a hypothesis on some parameters needs to be imposed. Let $f = \mathbb{E}[X_i]$, i.e, the possibility that at least one honest party solves POW at a given round.

Definition 8 (Honest Majority Assumption)**.** *With respect to the number of adversaries t out of the total number of parties n the following inequality holds: $\frac{t}{n-t} \leq (1-\delta)$, where $3f + 3\epsilon < \delta \leq 1$.*

The first inequality encapsulates the fact at least half of the parties are honest. The second inequality will be essential in order to obtain an important bound. In essence it represents a trade-off between f and ϵ. ϵ can be seen as the *quality of concentration of random values* in a run since it assures the number of successful and uniquely successful rounds is not far from their expected value and the number of adversary successful rounds is not much higher than its expected value. Particularly, it implies: $f, \epsilon < \frac{\delta}{3} \leq \frac{1}{3}$ and

$$(1+\epsilon)(1+f) < (1-\epsilon)(1-f)(1+\delta) < (1-\epsilon)(1-f)/(1-\delta)$$

For the first inequality $(1+\epsilon)(1+f) = 1+f+\epsilon+f\epsilon$ and $(1-\epsilon)(1-f)(1+\delta) = (1-\epsilon-f+f\epsilon) + ((1-\epsilon)(1-f)\delta)$ hence, the inequality will hold if, and only if, $2\epsilon + 2f < (1-\epsilon)(1-f)\delta$. As $2\epsilon + 2f = 2(\epsilon+f) < \frac{2\delta}{3}$ if the following inequality is true so will be the last $\frac{2\delta}{3} < (1-\epsilon)(1-f)\delta \iff \frac{2}{3} < (1-\epsilon)(1-f) \iff 0 < (1 - 3(\epsilon - f) + 3f\epsilon)$ but $(1 - 3(\epsilon - f) + 3f\epsilon) > 1 - \delta + 3f\epsilon > 3f\epsilon > 0$. For the second inequality $1 + \delta < \frac{1}{1-\delta} \iff 1 - \delta^2 < 1$, which is true.

Now some bounds for the random variables expectations will be calculated and they will be used in order to calculate other bounds that are true in the context of a typical execution. $X_i \rightsquigarrow B(1, f)$ and $f = 1 - (1-p)^{q(n-t)}$ where p is the probability of a successful query. This is because there are $(n-t)$ honest parties, each of which has q queries to the hash function every round, thus the probability of failing all of them is $(1-p)^{q(n-t)}$ and the probability of succeeding in one of them is the complementary event. On the other hand, $p = T/2^\kappa$ since out of all the possible outputs of the hash function only the smallest T are valid. The following bounds on $\mathbb{E}[X_i]$ hold:

$$(1-f)pq(n-t) < f = \mathbb{E}[X_i] = 1 - (1-p)^{q(n-t)} < pq(n-t)$$

For the first inequality $e^r x \geq (1+x)^r$ for reals x and $r \geq 0$ was used:

$$\frac{f}{1-f} = \frac{1-(1-p)^{q(n-t)}}{(1-p)^{q(n-t)}} = (1-p)^{-q(n-t)} - 1 > e^{-q(n-t)-1} - 1 \geq q(n-t) > pq(n-t)$$

For the second inequality Bernouilli's inequality was used $(1+x)^\alpha > 1 + \alpha x$ for reals $x > -1$ and $\alpha > 1$. Multiplying by -1 each member $-(1+x)^\alpha < -1 - \alpha x$:

$$1 - (1-p)^{q(n-t)} < 1 - 1 - (-p)q(n-t) = pq(n-t)$$

The Bernouilli inequality was used with $x = -p$ and $\alpha = q(n-t)$.

The following bounds link Y_i and Z_i expectations to f. Related to $\mathbb{E}[Y_i]$:

$$\mathbb{E}[Y_i] \geq q(n-t)p(1-p)^{q(n-t)-1} > pq(n-t)[1-pq(n-t)] > f(1-f) > \left(1 - \frac{\delta}{3}\right) f$$

Since Y_i is also a binomial distribution that models the probability that at a given round exactly one honest party solves POW its expectation will be at least the probability of all honest parties making all q queries and failing all of them except one, hence the first inequality. For the second inequality, first a unity was added to the exponent, since $(1-p) < 1$ this makes for the strict inequality and second Bernouilli's inequality was used. For the third inequality:

$$\left.\begin{array}{c}\text{Function } x \mapsto x(1-x) \text{ is increasing in } (0, 1/2) \\ f < pq(n-t) < f(1-f)^{-1} \underbrace{<}_{f < \frac{\delta}{3}} \frac{\delta}{3}(1-\frac{\delta}{3})^{-1} \underbrace{\leq}_{\delta \leq 1} \frac{1}{2}\end{array}\right\} pq(n-t)[1-pq(n-t)] > f(1-f)$$

Finally, for the last inequality $1 - f > 1 - \delta$ was used. Note that the fact that $f < 1/3$ is actually essential since function $x \mapsto x(1-x)$ is not monotone(much less increasing) in any interval (a, b) with $0 \leq a \leq 1/2 \leq b$.

Lastly, since Z_i models number of POW obtain by the adversary in a given round $Z_i \rightsquigarrow B(qt, p)$. Consequently, $\mathbb{E}[Z_i] = pqt$. Related to $\mathbb{E}[Z_i]$:

$$\mathbb{E}[Z_i] = pqt = pq(n-t)\frac{t}{n-t} < \frac{f}{1-f} \cdot \frac{t}{n-t} < \left(1 + \frac{\delta}{2}\right) \cdot f \cdot \frac{t}{n-t}$$

The first inequality is a consequence of the previous bounds and the second one is a consequence of: $\frac{1}{1-f} < \frac{1}{1-\frac{\delta}{3}} = 1 + \frac{\delta}{3-\delta}$ and $1 + \frac{\delta}{3-\delta} \leq 1 + \frac{\delta}{2} \iff \frac{2}{3-\delta} \leq 1 \iff \delta \leq 1$.

Now the previous bounds will be used to achieve useful bounds based on the properties of a typical execution:

Lemma 1. *In a typical execution under the honest majority assumption, for any set S of at least λ consecutive rounds it holds:*

(a) $(1-\epsilon)f|S| < X(S) < (1+\epsilon)f|S|$.

(b) $(1-\frac{\delta}{3})f|S| < (1-\epsilon)f(1-f)|S| < Y(S)$.

(c) $Z(S) < \frac{t}{n-t} \cdot \frac{f}{1-f}|S| + \epsilon f|S| \leq (1-\frac{2\delta}{3})f|S| < (1-\frac{\delta}{2})X(S)$.

Proof.

(a) It is enough with $X(S) = \sum_{s \in S} X_r \Rightarrow X(S) \rightsquigarrow B(|S|, f)$, since $X_s \rightsquigarrow B(1, f), \forall s \in S$ and, therefore $\mathbb{E}[X(S)] = |S|f$.

(b) $\mathbb{E}[Y(S)] = \mathbb{E}[Y]|S|$ and $Y(S) > (1-\epsilon)\mathbb{E}[Y(S)] = (1-\epsilon)\mathbb{E}[Y]|S| > (1-\epsilon)f(1-f)|S| > (1-\epsilon)f(1-f)|S| = (1-\epsilon-f-f\epsilon)f|S| > (1-\frac{\delta}{3}+f\epsilon)f|S| > (1-\frac{\delta}{3})f|S|$. The rest follows from the bounds given at the beginning.

(c) As with the other random variables $\mathbb{E}[Z(S)] = \mathbb{E}[Z]|S|$. The first inequality is obtained substituting $\mathbb{E}[Z(S)]$ for its value and using one of the bounds for $\mathbb{E}[Z]$. The second inequality is obtained . The las inequality follows from the lower bound obtained in (a): $(1-\frac{2\delta}{3})f|S| < (1-\frac{2\delta}{3}) \cdot \frac{1}{1-\epsilon} \cdot X(S) < (1-\frac{2\delta}{3}) \cdot \frac{1}{1-\frac{\delta}{3}} \cdot X(S) = \left(\frac{3-2\delta}{3-\delta}\right) X(S) = \left(1-\frac{\delta}{3-\delta}\right) X(S) < \left(1-\frac{\delta}{2}\right) X(S)$.

□

Remark 7. From Lemma 1 it can be deducted that tuning f value is a difficult task. $f = \mathbb{E}[X] = 1-(1-p)^{q(n-t)}$, thus, it is a function of $q = T/2^{\kappa}$ which is a function of the security parameter κ that can be changed during the course of the execution. The problem is, if f is "big", then the concentration of random variable $X(S)$ will grow but $Y(S)$ will get hurt since $Y(S)$ depends on $1-f$ also the amount of adversary success $Z(S)$ will grow too. Since the adversary is supposed to have rushing capabilities this goes to his advantage and ends up causing him to make a bigger contribution on the chain damaging *chain quality property* which will be soon defined. More in Remark 10. On the other hand, if f is very "small", then the overall amount of successful queries will be small, including the adversary's. But this is also a problem, if successful queries are not frequent enough then the chain will not make progress and no new content is added to the chain which will get stuck hurting another property that will be soon defined known as *chain growth property*. Thus, tuning f is a compromise. In the Bitcoin network, such calibration takes places every 2016 blocks and attempts to keep f in between $2-3\%$.

Corollary 1. *In a typical execution under the honest majority assumption, for any set S of at least λ consecutive rounds it holds: $Z(S) < Y(S)$.*

Proof. $Z(S) < (1-\frac{2\delta}{3})f|S| < (1-\frac{\delta}{3})f|S| < Y(S)$ □

Remark 8. Corollary 1 is important because it states that there is no way for the adversary $\mathcal{A}$ to perpetrate a successful *selfish mining* attack any a set of S or more consecutive rounds of a typical execution under the honest majority assumption. A selfish mining attack is that in which a party attempts to withold one or more successfully calculated blocks from being broadcast to the rest of the parties. This is one of the most feared attacks in Blockchain literature and the only used in the naive analysis made by Nakamoto in his famous article [Nak08]. Not only the adversary cannot win the race but it will clearly lose since only uniquely successful rounds are taken into consideration and successful rounds include uniquely successful rounds, thus, $Z(S) < Y(S) < X(S)$.

Lemma 2. *In a typical execution, any $k \geq 2\lambda f$ consecutive blocks of a chain have been computed in more than $k/2f$ consecutive rounds.*

Proof. Assume there is a set of consecutive rounds S' in which K blocks were computed and assume $|S'| < \frac{k}{2f}$. Adding rounds to S' makes it so the same amount of blocks or more are computed, as consequence let S be a set of consecutive rounds with $|S| = \lceil \frac{k}{2f} \rceil + 1$ hence $X(S) + Z(S) \geq k$. But this is a contradiction since

$$X(S) + Z(S) < (2 + \epsilon - \frac{2\delta}{3})f|S| \leq (2 - 2f)f|S| \leq (1 - f)(k + 4f) < k$$

For the first inequality the bounds from Lemma 1 (a) and (b) were used[1]. For the second inequality the honest majority assumption was used $3f + 3\epsilon < \delta \Leftrightarrow 3\epsilon - \delta < -3f \Longrightarrow 2 + \epsilon - \frac{2\delta}{3} = 2 + \frac{3\epsilon - 2\delta}{3} < 2 - \frac{3f+\delta}{3} < 2 - \frac{3f+3f}{3} = 2 - 2f$. For the third inequality $|S| = \lceil \frac{k}{2f} \rceil + 1 \leq \frac{k}{2f} + 2$. Finally the last inequality can be checked directly $(1 - f)(k + 4f) < k \Leftrightarrow K + 4f - fk - 4f^2 < k \Leftrightarrow 4f < fk + 4f^2 \Leftrightarrow 4 < k + f^2$ and the last inequality is true since $\frac{k}{2f} \geq \lambda \geq 2/f \Rightarrow k \geq 4$. □

Remark 9. Lemma 2 basically states that in a typical execution apply for every set of $3\lambda f$ or more consecutive blocks. Given a chain C, at first glance, no one can tell how many rounds were needed for a subset of blocks to be calculated this is a problem since it will be necessary to apply Lemma 1 in order some relevant properties. Lemma 2 is very important because it offers a one way relation between round number and consecutive block number making it feasible to use properties of a typical execution in a big enough subset of consecutive blocks which, contrary to rounds, can be counted just by looking at the chain.

5.2 Chain Growth Property

The first property to be defined is the *common prefix property* parameterized by $s \in \mathbb{N}$ and $\tau \in (0, 1)$. This property states that in a fixed amount of rounds all honest parties chains will grow proportionally to the number of rounds considered, no matter what the adversary does. This is very important in order to assure chains make progress and, thus, new content keeps being added to the chain.

[1] Lemma 1 can be used since $\lceil \frac{k}{2f} \rceil \geq \lambda$

Definition 9 (Chain Growth Property)**.** The *chain growth* property with parameters $\tau \in (0, 1)$ and $s \in \mathbb{N}$ states that for any honest party P that has a chain $\mathcal{C}$ in $\texttt{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$, it holds that after any s consecutive rounds it adopts a chain that is at least $\tau \cdot s$ blocks longer than $\mathcal{C}$.

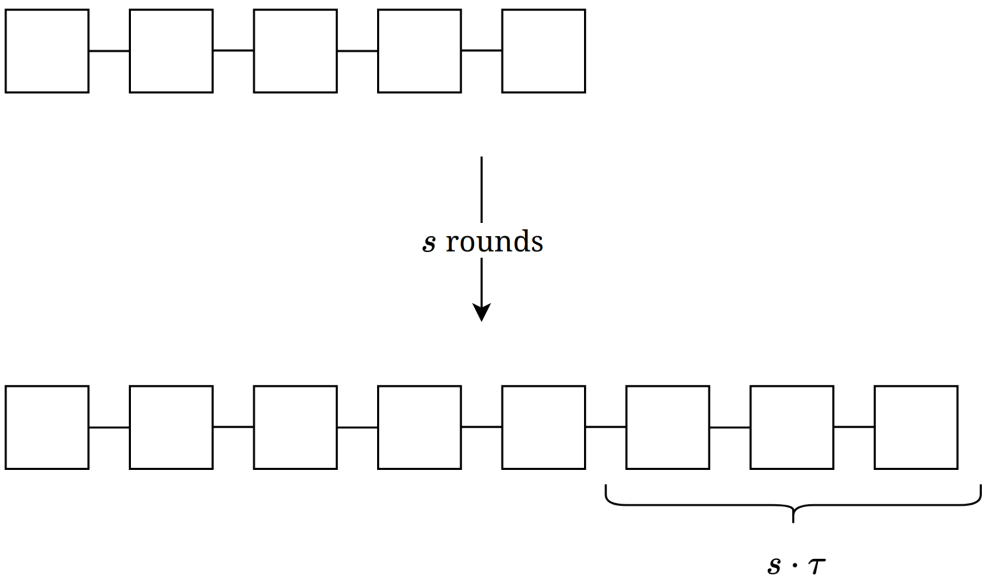

Figure 5.1: Chain growth property sketch

The following lemma states that, at any given round, the chain length of any honest party will be at least the sum of the number of successful rounds. This implies that the growth of the honest parties' chains will be at least the rate of the successful rounds independently of what the opponent does.

Lemma 3 (Chain Growth Lemma)**.** *Suppose that at round r an honest party has a chain of length l. Then, by round $s \geq r$, every honest party has adopted a chain of length at least $l + \sum_{i=r}^{s-1} X_i$.*

Proof. By induction on $s - r \geq 0$. For the base case $s = r$, if an honest party has a chain C of length l, then that chain C must have been broadcast at a round earlier that r, thus, all honest parties must have received C. For the inductive step, by the inductive hypothesis by round $s - 1$ every honest party has received a chain C of length $l + \sum_{i=r}^{s-2} X_i$. At round s, if $X_{s-1}(S) = 0$ there is nothing to do. If $X(S)_{s-1} = 1$, then some honest party achieved PoW and hence, broadcast a chain of length $l' + 1$, but since $l' = l + \sum_{i=r}^{s-2} X_i \Rightarrow l' + 1 = l + \sum_{i=r}^{s-1} X_i$. □

Theorem 2 (Chain Growth)**.** *In a typical execution the chain growth property holds with parameters $\tau = (1 - \epsilon)f$ and $s \geq \lambda$.*

Proof. Suppose that at round r an honest party has a chain C of length l. Then, by round, s it will have a chain of length, at least, $l + X(S)$ where S represents the s rounds following

r, i.e, $S = \{i : r \leq i < r + s\}$. Now, $|S| \geq \lambda$ by hypothesis and Lemma 1(a) applies $X(S) > \underbrace{(1-\epsilon)f}_{\tau} \cdot \underbrace{|S|}_{s}$. □

5.3 Common Prefix Property

The second property considered is the *common prefix property* parameterized by $k \in \mathbb{N}$. This property states that given an honest node chain any honest node's chain in a later round will share at least the first k blocks with it. This property is very important because it assures that the parties are working on a common substratum that will never be altered, thus, their work is not in vain.

Definition 10 (Common Prefix Property)**.** The *common prefix* property with parameter $k \in \mathbb{N}$ states that for any pair of honest players P_1, P_2 adopting the chains $\mathcal{C}_1, \mathcal{C}_2$ at rounds $r_1 \leq r_2$ in $\texttt{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$ respectively, it holds that $\mathcal{C}_1^{\lceil k} \preceq \mathcal{C}_2$

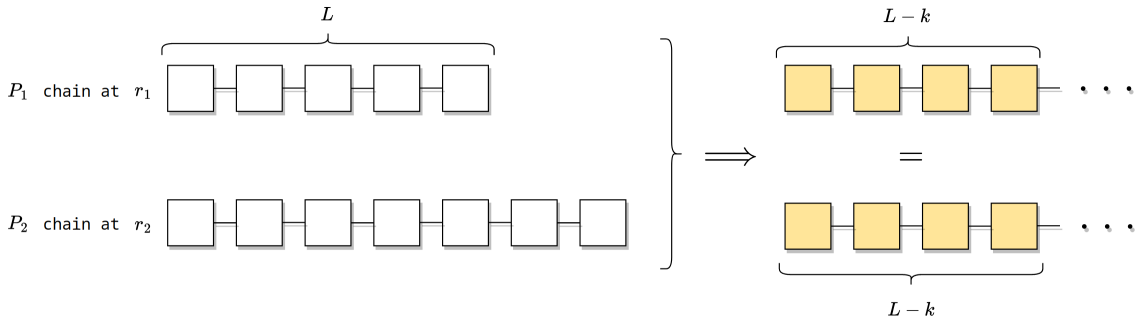

Figure 5.2: Chain common prefix sketch

Definition 11 (Block types)**.** A block is said to be *honest* if it was **computed and diffused** by an honest party and is said to be *adversary* if it was **diffused** by the adversary.[2]

Lemma 4. *Suppose the k-th block B of a chain $\mathcal{C}$ was computed by an honest party in a uniquely successful round. Then the k-th block a chain $\mathcal{C}'$ either is B or has been computed by the adversary.*

Proof. Suppose an honest block B' is the k-th block in an honest party's chain C'. Also, suppose B' is different from B. Let r be the round when B was calculated. Since r is a uniquely successful round B' had to be computed at a different round. But it cannot be computed at a later round: By Lemma 3, at the beginning of r, every honest party's chain's length was the same $k-1$. Since r is a uniquely successful round, during r only one honest party calculated a new block and then broadcast its extended chain of length k. Every honest block calculated at a later round will be extending a chain of length more than k. It also couldn't be computed before: since a block is calculated upon a previous one

[2]This definition avoids controversy in demonstration of Lemma 5

if an honest party calculated a block in extending the $k-1$-th block of its chain it would broadcast it at the end of the round and all blocks calculated at a later round would be extending a chain of length k, particularly at round r. This contradicts the hypothesis "B is the k-th block of a chain at round r". □

Lemma 5 (Common Prefix Lemma)**.** *Suppose that at round r of a typical execution an honest party has a chain C_1 and an honest party adopts a chain C_2 of length at least $len(C_1)$, then $C_1^{\lceil k} \preceq C_2$ and $C_2^{\lceil k} \preceq C_1$ for every $k \geq 2\lambda f$.*

Proof. The proof is by contradiction. Assume that, in a typical execution, for some $k \geq 2\lambda f$, either $C_1 \not\preceq C_2$ or $C_2 \not\preceq C_1$. Let $r*$ be the round at which the last honest block of C_1s and C_2s common prefix was computed. If no such block exists let $r* = 0$. Let $S = \{i : r* < i < r\}$, i.e, the set of rounds between $r*$ and r. It will be proved that

$$Z(S) \geq Y(S)$$

In order to prove it each block calculated in a uniquely successful round will be paired with an adversary block.

First note that if the block mined at $r*$ occupies position l then every block calculated in S must extend a chain of length at least l. Now, for each uniquely successful round $u \in S$ let j_u be the chains position of the block calculated at round u. Consider $P = \{p_u : u \text{ is uniquely succesful in } S\}$. Note that at round r both chains have length at least $max(P)$. Therefore, for each $j \in J$ there is a block in position p of both chains. Note that $P \neq \emptyset$ since trivially $Z(S) \geq 0$. Now it will be proved that for each $p \in P$ at least one of the blocks in position p is an adversary block.

- p is on the common prefix: for being in the common prefix both block are equal, if the block was honest it would contradict $r*$ definition since every block calculated in S is calculated after $r*$ but then, as noted before, there would exist a common honest block in position $j > l$, hence the block calculated in $r*$ would not be the last common honest block.

- p is not on the common prefix: if there are two honest blocks in position p then, by Lemma 4 they must be equal, furthermore, there would be a common prefix of length p since when the block is broadcast the complete chain is broadcast with it. This a contradiction with p not being in the common prefix.

Last, by $C_1 \not\preceq C_2$ or $C_2 \not\preceq C_1$ for some $k \geq 2\lambda f$, $len(C_1) - l > k$ or $len(C_2) - l > k$, in any case, $l > 2\lambda f$ and, by Lemma 2, the properties of a typical execution apply. Particularly, by Corollary 1, $Z(S) < Y(S)$. □

Theorem 3 (Common Prefix)**.** *In a typical execution the common prefix property holds with parameter $k \geq 2\lambda f$.*

Proof. Asume, towards a contradiction, chains C_1 and C_2 are adopted by parties P_1 and P_2 in rounds r_1 and r_2 such that $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$. By Lemma 5 in r_1 every honest parties chain contains $\mathcal{C}_1^{\lceil k}$, thus, there must be a round $r_1 \leq r \leq r_2$ in which an honest party P has a chain C containing $\mathcal{C}_1^{\lceil k}$ and adopts a another chain C' such that $\mathcal{C}_1^{\lceil k} \not\preceq C'$. Now, $C^{\lceil k} \not\preceq C'$ and $len(C') \geq len(C)$, contradicting Lemma 5. □

Note that, in the proof, P doesn't have to be P_2 since the C_2 may have been built by a variety of different parties.

5.4 Chain Quality Property

The last property considered is the *chain quality* parameterized by $\mu \in (0,1)$ and $l \in \mathbb{N}$. This property states that at least a portion μ in any subset of length l of consecutive blocks of any honest party's chain is made up of honest blocks. This property is important since it assures the contribution of the honest parties to the chain is high which in turn means that the impact of the adversary is limited.

Definition 12 (Chain Quality Property)**.** The *chain quality* property with parameters $\mu \in (0,1)$ and $l \in \mathbb{N}$ states that for any honest party P with chain $\mathcal{C}$ in $\texttt{VIEW}^{t,n}_{\Pi,\mathcal{A},\mathcal{Z}}$, it holds that for any l consecutive blocks of $\mathcal{C}$ the ratio of honest blocks is at least μ.

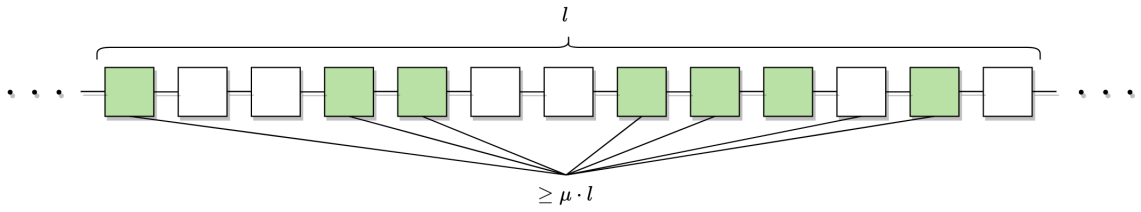

Figure 5.3: Chain quality property sketch

Theorem 4 (Chain Quality)**.** *In a typical execution the chain quality property holds with parameters* $l \geq 2\lambda f$ *and* $\mu = 1 - (1+\frac{\delta}{2}) \cdot \frac{t}{n-t} - \frac{\epsilon}{1-\epsilon} > 1 - (1+\frac{\delta}{2}) \cdot \frac{t}{n-t} - \frac{\delta}{2}$.

Proof. Let B_k be the k-th block of a chain C of an honest party P at some round r so that $C = B_1 \ldots B_l en(C)$ and consider l consecutive blocks $B_i, \ldots, B_j$. Let L be the smallest set of consecutive rounds that include the l given blocks (i.e $i' \leq i$ and $j \leq j'$) such that:

1. B_i was computed by an honest party. If no such block exist the genesis block B_1 is taken.

2. There exists a round at which an honest party was trying to extend the chain ending at block B_j. This is well defined since $B_{len(C)}$ is always at the head of a chain an honest party is trying to extend.

Let r_1 be the round at which $B_{i'}$ was created ($r = 0$ for (B_1)), let r_2 be the first round an honest party tried to extend $B_{j'}$ and let $S = \{r : r_1 \leq r \leq r_2\}$.

Take μ as in the statement and let x be the number of honest blocks in the l selected blocks. Towards a contradiction assume that

$$\mu L \geq \mu l > x$$

Suppose that the L blocks were computed during the rounds in S. Then

$$Z(S) = L - x > (1-\mu)L \geq (1-\mu)X(S) = \left(\left(1+\frac{\delta}{2}\right)\cdot\frac{t}{n-t} + \frac{\epsilon}{1-\epsilon}\right)X(S)$$

The first inequality comes from $-x > -\mu L$. For the second inequality, suppose $X(S) > L$. At round r_1 an honest party produced B_i' and therefore has a chain of length i'. By Lemma 3, at round r_2 every honest party will have a chain of length $i' + X(S) > i' + L = j' + 1 > j'$ which is a contradiction. Finally, by Lemma 2 $|S| \geq \lambda$ and the bounds of a typical execution apply, hence

$$Z(S) \geq \left(\left(1+\frac{\delta}{2}\right)\cdot\frac{t}{n-t} + \frac{\epsilon}{1-\epsilon}\right)X(S) \geq \left(1+\frac{\delta}{2}\right)\cdot\frac{t}{n-t}\cdot X(S) + \epsilon f|S| > Z(S)$$

The second inequality uses Lemma 1 (a) and the third inequality uses Lemma 1 (c). The contradiction comes from supposing all blocks in L have been computed during the rounds in S and the relation between x and L.

If L contains blocks calculated in rounds not in S. By the way L is defined these blocks must have been computed by the adversary. If a block was calculated before S then in order to belong to L there would need to be a copy or a prediction since, the alternative would be the adversary building a chain of length bigger than i' before the round $B_{i'}$ was calculated, but this would contradict $B_{i'}$ definition since when the adversary released its chain the block at position i would be necessarily adversary. If a block was calculated after S but still belongs to L then there would need to be an insertion since otherwise the adversary would have to attempt to substitute one of the blocks in L by outgrowing the honest chain beginning at any position in L, but this is a contradiction with the definition of $B_{j'}$ since when such chain was broadcast the last block in L would not comply with its requirement of being worked on an extension by an honest node. As seen any of the possibilities requires a disruptive event that do not occur in a typical execution. The contradiction comes from assuming a block in L has been computed in a round not in S and the relation between x and L.

Adding both facts there is a contradiction with the relation between x and L which proves the theorem. □

Corollary 2. *Any $\lceil 2\lambda f \rceil$ consecutive blocks in the chain of an honest party during a typical execution contain at least one honest block.*

Proof. $\mu = 1-(1+\frac{\delta}{2}\cdot\frac{t}{n-t}-\frac{\epsilon}{1-\epsilon}) \underbrace{>}_{\epsilon < \frac{\delta}{3}} 1-(1+\frac{\delta}{2})\cdot\frac{t}{n-t}-\frac{\delta}{2} \underbrace{\geq}_{\frac{t}{n-t} \leq (1-\delta)} 1-(1+\frac{\delta}{2})\cdot(1-\delta)-\frac{\delta}{2} =$

$1-[1+\frac{\delta}{2}-\delta-\frac{\delta^2}{2}]-\frac{\delta}{2} = \frac{\delta^2}{2} > 0$ □

Remark 10. It can argued(informally) that Theorem 4 is tight under the hypothesis that chain ties always favor the adversary. This lies in the description of function `maxValid` and the fact that, in the considered model, the adversary has rushing capabilities, thus, they can get to write their chain in the input tape of every party before any honest party gets to diffuse its message. When the number of parties is large, in a round where an honest party finds a solution and the Adversary too(or had already found one) all other honest parties

will prefer the adversary's chain and the effect of a single honest party opting for its own block is negligible.

The attack that is going to be described a type of selfish mining attack that reaches the stated bound. Initially, the adversary works on the same chain as every honest party. However, when it finds a solution it keeps it to itself and continuous to extend its private chain. When an honest party finds a solution the adversary makes use of their rushing capabilities in order to release the first private block from their chain causing al honest parties to adopt it. This strategy maximizes the adversarial blocks in the Blockchain up to the upper bound of Theorem 4.

Consider a set S of consecutive rounds. With constant probability, in this set of rounds, the adversary will obtain, at least $z \geq pqt|S|$ solutions, while the honest parties will obtain at least $pq(n-t)|S| > x$ solutions. By following the detailed strategy the adversary can manage to counter(with overwhelming in n) z out of x honest blocks. Now, for the smallest sequence of blocks that contains the z adversary blocks: $\mu \geq \frac{(x-z)}{x} = \frac{(n-2t)}{(n-t)} \geq 1 - (1 + \frac{\delta}{2}) \cdot \frac{t}{n-t} - \frac{\epsilon}{1-\epsilon}$ for any $\delta, \epsilon \in (0, \frac{1}{3})$.

From the above, in order to improve on chain quality a more favorable(for honest parties) tie breaking criteria needs to be considered. This translates in a need for a better connected network in which the adversary doesn't have such an strong capability as rushing or it is diminished in one way or another.

Chapter 6

Applications

This chapter has two main sections. The first one is comprised by the formal description of the Public Transaction Ledger problem as well as the description of a solution for such problem. The second one is comprised by the formal description of the General Byzantines problem and two posible three solutions for such problem. All but one of the provided solutions make extensive use of the properties of the protocol proved in Chapter 5. This chapter is mainly based on [GKL15], [San06] and [Zha13].

6.1 Public Transaction Ledger

A ledger $\ell \in \mathcal{L}$ is a sequence of sequences of transactions tx $\in \mathcal{T}$. The blocks content will be sequences of transactions $\langle tx_1, \ldots, tx_m \rangle$. Thus, by definition, the chains content $x_{\mathcal{C}} = \langle x_1, \ldots, x_n \rangle$ is a ledger. The *position* of a transaction in a chain is determined by its blocks index along with its sequence index, i.e, a transaction tx has position (i, j) if it is the j-th transaction of the i-th block.

A transaction relates the state of one or more *accounts* $a_1, a_2, \ldots$ etc. An account is an object that encapsulates a state. In order to manage accounts and relations between transactions two functions need to be considered. The following external functions need to be defined by the application that wishes to implement any public transaction ledger:

- `Txgen`$(\cdot)$: A functionality. It needs to provide an answer to the following queries:
  - `GenAccount`: It generates an account a.
  - `IssueTx`$(\hat{\text{tx}})$: It returns a transaction tx given that $\hat{\text{tx}}$ is some properly provided string.
- $C(\cdot, \cdot)$: a symmetric relation on $\mathcal{T}$. Returns true when two transactions are conflicting.

A ledger is said to be valid if it contains only valid transactions. For the sake of simplicity no more restrictions will be imposed on the ledgers but the results hold for further restricted ledgers. Functionality `Txgen` is said to be unambiguous if it holds that for all PPT $\mathcal{A}$ produces a transaction tx' such that $C(tx', tx) = 1$ for a tx issued by `Txgen` is negligible in κ. I.e, when the adversary doesn't have the possibility to create transaction that are conflicting with the ones generated by `Txgen` for honest accounts.

For a protocol to implement a robust public transaction ledger it needs to satisfy (i) *Persistence* and (ii) *Liveness*. Persistence property implies that once an honest party reports a transaction "deep enough" in its local chain, from that point on, all honest parties will report such transaction at exactly the same position in the ledger. In this sense, Persistence means that all honest parties agree on all transaction and the order at which they took place. Liveness implies that any transaction that comes from `Txgen`, when provided by the environment to all honest players for a sufficient number of rounds, will eventually be inserted in all parties ledgers.

Definition 13 (Public Transaction Ledger Problem)**.** *A protocol Π implements a robust public transaction ledger in the q-bounded synchronous setting if it organizes the ledger as a hashchain of blocks of transactions and it satisfies:*

(i) Persistence: Parameterized by $k \in \mathbb{N}$, if in a certain round an honest player reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger, then tx will be reported by any honest player in the same position in the ledger, from this round on.

(ii) Liveness: Parameterized by $t, k \in \mathbb{N}$, provided that a transaction issued by `Txgen` is given as input to all honest players continuously for t consecutive rounds, then all honest parties will reports this transaction more than k blocks from the end of the ledger.

Related to Persistence, parameter k will be known as the "depth" parameter and transactions allocated on the first k blocks of the chain will be called *stable*. The smaller k, the sooner transactions become stable, thus, it is desirable for k to be as small as posible. It is important to note that it cannot be demanded for a transaction to be reported as stable the same rounds it is reported to be on the first k blocks of a given chain. This is because the adversary could advance the chain of only a few parties leading to a disagreement that will last until next round takes place.

Related to Liveness, parameter t will be known as the "wait time" parameter and k will also be known as the "depth" parameter. In this particular model transaction broadcast is handled by the environment Z it is required that the environment Z provides each transaction as input so that the liveness property guarantees its eventual inclusion to the ledger. An alternative definition of the liveness property can be made so that transaction exchange between parties is possible(or even mandatory). It that case, a single honest party receiving the transaction as input would suffice.

6.1.1 Nakamoto Solution

Nakamoto provided a protocol that implements a public transaction ledger. Such protocol will be referred to as Π_{PL} and is specified in Table 6.1. In order to prove that such protocol implements a public transaction ledger persistence and liveness properties will be proved.

Function	Parameters	Description
$V(\cdot)$	-	$V(\langle x_1, \ldots, x_n \rangle)$ is the if and only if sequence $\langle x_1, \ldots, x_n \rangle$ is a valid ledger.
$R(\cdot)$	-	$R(\mathcal{C})$ returns $\langle x_1, \ldots, x_n \rangle$.
$I(\cdot)$	-	If the input tape contains (INSERT, v), it parses v as a sequence of transactions and retains the largest subsequence x' that is valid with respect to $x_{\mathcal{C}}$ and whose transactions are not included in $x_{\mathcal{C}}$.

Table 6.1: Protocol Π_{PL}

Lemma 6. *(Persistence) Under the Honest Majority Assumption, it holds that Π_{PL} described in Table 6.1 parameterized with $k = \lceil 2\lambda f \rceil$ satisfies Persistence with probability at least $1 - e^{-\Omega(\epsilon^2 \lambda f)}$.*

Proof. The result follows directly from the Common Prefix property. Consider a typical execution and let $\mathcal{C}_1$ be the local chain of an honest party P_1 on round r_1. Suppose in round r_1 tx is a transaction that belongs to $\mathcal{C}_1^{\lceil k}$. Consider a local chain $\mathcal{C}_2$ belonging to an honest party P_2 on round $r_2 \geq r_1$. By the common prefix property, $\mathcal{C}_1 \preceq \mathcal{C}_2$ and the statement follows. □

Lemma 7 (Liveness)**.** *Under the Honest Majority Assumption, it holds that Π_{PL} described in Table 6.1 parameterized with $t = \lceil 4\lambda/(1-\epsilon) \rceil$ rounds and $k = \lceil 2\lambda f \rceil$ satisfies Liveness with probability at least $1 - e^{-\Omega(\epsilon^2 \lambda f)}$.*

Proof. It will be proved that, assuming all honest players receive as input the transaction tx for at least t rounds, then, in a typical execution there exists an honest party with chain $\mathcal{C}$ such that tx is included in $\mathcal{C}^{\lceil k}$. In a typical execution, after t rounds the honest parties chain will be at least $4\lambda/(1-\epsilon) = 2k$ blocks longer. At that exact moment, by the chain quality property, the length-k prefix of any honest parties pruned-k local chain $\mathcal{C}^{\lceil k}$ contains at least one block mined by the honest parties. Such block must contain tx since it is impossible for adversarial $\mathcal{Z}, \mathcal{A}$ to produce a conflicting transaction tx'. Such block must be equal in all chains by the Persistence property. □

Theorem 5. *Under the Honest Majority Assumption, it holds that Π_{PL} described in Table 6.1 parameterized with $t = \lceil 4\lambda/(1-\epsilon) \rceil$ rounds and $k = \lceil 2\lambda f \rceil$, implements a public transaction ledger with probability at least $1 - e^{-\Omega(\epsilon^2 \lambda f)}$.*

Bitcoin Ledger

In this context protocol parties will be referred to as miners. Transactions and accounts are defined respect to a digital signature scheme. A digital signature scheme is a functionality that allows to reproduce the properties of a physical signature digitally, namely, authentication and non-repudiation. The considered scheme is comprised of tree algorithms $\langle$`KeyGen`, `Sign`, `Verify`$\rangle$. `KeyGen` provides a pair (pk, vk) composed by a unique private key and public key, `Sign` signs any provided message using a provided private key and `Verify` verifies a given message

has been signed by a given account. A signature scheme is *existentially unforgeable* when adversaries can't create signatures that verify for messages they have not already a signature for.

An account is a pair $a = (publicKey, G(publicKey) = adress)$ where $G(\cdot)$ is a hash function and $publicKey$ is also referred to as the account address since it identifies an account in an explicit manner.

A transaction tx is a structure $\{a \rightarrow ((vk, \sigma), \{(a_1, b_1), \ldots, (a_n, b_n)\})$ where a is the account to be debited, $a_1, \ldots, a_n$ are the accounts to be credited with funds $b_1, \ldots, b_n$ and (vk, σ) is a pair of verification key and digital signature for the message $\{a \rightarrow (\{(a_1, b_1), \ldots, (a_n, b_n)\})$. Outgoing balance will be referred to as a transaction output and incoming balance will be referred to as a transaction input.

`Txgen` oracle has to be specified. It's job will be generating transactions in behalf of honest users.

- `GenAccount`: Calls `KeyGen` and obtains a pair (pk, vk) then computes vk hash to obtain account $a = (vk, G(vk))$ which is returned. The private key pk is kept in the the state of `Txgen`.

- `IssueTrans`($\hat{tx}$): Returns a transaction tx as long as $\hat{tx}$ is a transaction that only misses the signature of an account maintained by `Txgen`.

In order to assure the system works correctly it is necessary to impose that each account is only allowed a single transaction, thus, $C(tx_1, tx_2) = 1$ if and only if $tx_1 \neq tx_2$ and tx_1, tx_2 have share the input account. Under this conditions it can be proved that:

Proposition 1. *Assume that $\langle Keygen, Sign, Verify\rangle$ is an existentially unforgeable signature scheme. Then oracle `Txgen` is unambiguous.*

Proof. Since the signature scheme is existentially unforgeable there is no way for adversary to forgery a transaction unless such transaction has already occurred in the past but this cannot happen since $C(\cdot, \cdot)$ doesn't allow more than one transaction from any given account. □

The fact that each account is only able to perform one transaction is not restrictive since one transaction may have more than one output, thus, if a user desires to spend only a fraction of their credit its wallet may create a new account in which the remaining part of their credit can be deposited avoiding any loss.

Transaction validity needs to be determined in order to define the valid ledgers in the Bitcoin environment. A transaction `tx` is said to be valid if all signatures verify and involved debited accounts balance is greater that the sum of the credited amount, that is, every account $a \in tx$ verifies $\sum_{j=1}^{q} b_i \geq \sum_{j=1}^{t} b'_i$ where b_i is a unique that was previously(including this current transaction) given to a and b'_i is a unique output in `tx`. Calculating PoW means using computational power to calculate a hash function time after time, thus, miners expect to receive a reward for perpetrating such task. Miners receive a reward in the form of a *transaction fee* $= \sum_{j=1}^{q} c$ where $c = \sum_{j=1}^{q} b_i > \sum_{j=1}^{t} b'_i$ plus a flat reward r_m which depends on the length of the chain m that it the miner is trying to extend. Transaction fees can be specified as special transactions of the form "$\varnothing \rightarrow \ldots$". This mechanism allows the

Date reached	Block	BTC Reward	Year (Estimate)	BTC Added	Stage BTC Added
1/3/09	0	50	2009	2625000	12.5%
4/22/10	52500	50	2010	2625000	25.00%
1/28/11	105000	50	2011	2625000	37.50%
12/14/11	157500	50	2012	2625000	50.00%
11/28/12	210000	25	2013	1312500	56.25%
10/9/13	262500	25	2014	1312500	62.50%
8/11/14	315000	25	2015	1312500	68.75%
7/29/15	367500	25	2016	1312500	75.00%
	420000	12.5	2017	656250	78.13%
	472500	12.5	2018	656250	81.25%
	525000	12.5	2019	656250	84.38%
	577500	12.5	2020	656250	87.50%
	630000	6.25	2021	328125	89.06%
	682500	6.25	2022	328125	90.63%
	735000	6.25	2023	328125	92.19%
	787500	6.25	2024	328125	93.75%

Table 6.2: Bitcoin Reward Geometric Progression

spending accounts to grab the miners attention toward its transactions when needed since miners will try to maximize transaction fees. It is important to note that an account is not forced to give all its spare credit as a transaction fee since new accounts can be created at any given time with no added cost.

The initial reward for extending the Bitcoin chain was 50 BTC. The reward progression for Bitcoin follows the geometric progression in Table 6.2.

A sequence of transactions $\langle \varnothing \to \{(a_1, b_1)\}, tx_1, \ldots, tx_n \rangle$ is said to be valid with respect to a ledger $l = \langle x_1, \ldots, x_m \rangle$ if each transaction is valid with respect to the l extended by transactions previous to it, i.e, for all $j \in \{1, \ldots, n\}$ transaction tx has to be valid with respect to ledger $\langle x_1, \ldots, x_m, tx_1, \ldots, tx_{j-1} \rangle$.

The set of valid ledgers $\mathcal{L}$ with respect to a reward progression $\{r_j\}_{j \in \mathbb{N}}$ contains any ledger l which extends a ledger in $\mathcal{L}$ by a valid sequence of transactions. The first transaction sequence of any ledger $l \in \mathcal{L}$ contains a single reward transaction $\varnothing \to \{(a_1, b_1), \ldots, (a_n, b_n)\}$ that satisfies $\sum_{j=1}^{n} b_i = r_0$, where r_0 is the first initial reward. This initial rewards distributes the first coin credit among the ledger's initiator(s). In the case of Bitcoin it was supposedly Nakamoto who collected the first reward.

6.2 Byzantine Agreement

This problem is very close in essence to the entity-fault-tolerant consensus and is traditionally explained with a warlike analogy. Processes as represented as Lieutenants. The Lieutenants's objective is to conquest a fortress. There is a distinguished process, the General. Lieutenants can only communicate through messages. The General will send an order v to each other Lieutenant to attack or retreat. Some Lieutenants may be traitors(including the General) and may exhibit any behavior, i.e byzantine failures[1]. Non-traitor Lieutenants are called loyal.

Definition 14 (Byzantine Generals Problem)**.** *A protocol* Π *is said to solve the Byzantine Agreement(BA) problem if it satisfies the following properties:*

- ***Agreement****: There is a round after which all honest parties return the same value if queried by the environment.*
- ***Validity****: The output returned by an honest party* P *is the input of an honest party* P' *on round 1.*
- ***Termination****: Agreement is achieved in a finite time.*

Remark 11. *In the* q*-bounded computational model termination is actually guaranteed.*

The validity condition described in the problem is *strong validity*. *Weak validity* implies that if all parties begin with the same value then the output value has to be that input value. There are no conditions in case more two or more different input values are provided. This is only relevant in case there are more than two possible input values, in which case, the strong validity property implies weak validity and not the other way around. The described protocol has long been known to have solution in a deterministic model if and only if $n > |V|t$ where V is the input. The protocol that solves the problem is called the Oral Messages Protocol:

Lemma 8. *The Oral Messages protocol solves the problem for* $n > 3m$*.*

The following result is also true:

Lemma 9. *There is no protocol that satisfies Byzantine Consensus in an asynchronous system with* n *Liutenants and* m *traitors with* $n < 3m$*.*

Theorem 6. *The maximum number of failures any protocol can tolerate while achieving Byzantine Consensus is* $m = max\{f \in \mathbb{N} : n > 3f\}$*.*

Using these two lemmas proven in the famous article [PSL80] the theorem is proven.

Remark 12. *Note as stated before the Theorem only holds in a deterministic model. The model described in this work is probabilistic, thus the Theorem cannot be applied on it.*

6.2.1 Nakamoto Solution

This is an interpretation of the solution given by informally by Nakamoto in a mailing list [Nak].

The protocol is very simple. Blocks content is a bit value $x \in \{0, 1\}$. For a chain to be valid the content of all its blocks must be the same. If a party has no local chain it expects one and only one value in its input tape and will attempt to calculate a block with such value. If a party has a local chain it will try to extend it with a block containing the same value as the one present in its chain.

[1]The name Byzantine failure actually comes from this analogy. The Byzantine Empire is sometimes taken as a reference in political instability due to the plots of intrigue, betrayal and assassination that were said to take place in palace.

Intuitively, even if there are several times in which two or more parties extend their chain during the same round there will eventually be a uniquely successful round in which only one party will extend its chain. Thus, since Nakamoto supposed an advantage on the PoW achievement of honest parties over the Adversary, Agreement is achieved. While this is true and it will be demonstrated, Validity doesn't seem to be taken in consideration and it will be proved that this protocol doesn't achieve Validity(with high probability). Only environments $\mathcal{Z}$ that provide an initial value to all parties in P are considered.

Formally, external functions are defined as follows.

Function	Parameters	Description
$V(\cdot)$	-	$V(\langle x_1, \ldots, x_n \rangle)$ is true if and only if $x_1 = \ldots x_n \in \{0, 1\}$
$R(\cdot)$	k	If len($\mathcal{C}$) $\leq k$, $R(\mathcal{C})$ is the (unique) value that is contained in each block of chain $\mathcal{C}$. Otherwise it is undefined.
$I(\cdot)$	-	If $\mathcal{C} = \epsilon$ the functionality returns the value taken from the input tape INPUT() which is expected to contain (INSERT, v). Otherwise the (unique) value present in the $\mathcal{C}$ is returned.

Table 6.3: Protocol $\Pi_{\text{BA}}^{\text{nak}}$

First, Agreement well be proved. It follows from the common prefix property. If len($\mathcal{C}$) $> k$ then $R(\mathcal{C})$ will return the value present in the common prefix for every single party.

Lemma 10 (Agreement)**.** *Under the Honest Majority Assumption, it holds that protocol Π_{BA}^{nak} described in Table 6.3 parameterized with $k = \lceil 2f\lambda \rceil$ running for a total number of rounds $L \geq 2k/f$ satisfies Agreement with probability at least $1 - e^{-\Omega(\epsilon^2 f\lambda)}$.*

Proof. By the common prefix property, after L rounds, every honest party will have a chain of length at least k. This is because $\tau L = 2(1-\epsilon)k > k$ since $\epsilon < \frac{1}{3}$ by the Honest Majority Assumption. Since chains contain unique values a disagreement between honest parties chains would imply disjoint chains which contradict the common prefix property considering all chains have more than k blocks. □

Note that the inequality can be further fit by taking $L \geq 1.5k/f$.

Regarding Validity; Unless the adversary PoW success rate is negligible compared to the honest players, Validity cannot be guaranteed with overwhelming probability. This is because if the adversary finds a solution faster than the honest parties it will broadcast it and every party will attempt to extend a chain with a value that is not guaranteed to be among the initial ones for any honest party. Hence, it can be formally proved that Validity can only be assured with $(n-t)/n$ probability, which is the rate of honest parties running the protocol.

6.2.2 Alternative Solution

An alternative solution will be given using the backbone protocol that, under an strengthened version of Honest Majority, can assure Agreement and Validity as long as the adversary's hashing power is bounded by $\frac{1}{3}$. The protocol is based on Π_{BA}^{nak} with two major differences.

First, parties always try to add their input value to the current chain they are attempting to extend. Second, chains no longer contain unique values, after round L they output the majority of their local length k prefix. For the second condition to be correctly defined binary BA is considered. Just as before, the only environments $\mathcal{Z}$ considered are the ones that provide an initial value to all parties in P.

Formally, the protocol is defined as follows:

Function	Parameters	Description
$V(\cdot)$	-	$V(\langle x_1, \ldots, x_n \rangle)$ is true if and only if $v_1, \ldots, v_n \in \{0,1\}$.
$R(\cdot)$	k	If len($\mathcal{C}$) $> 2k$, $R(\mathcal{C})$ returns the majority bit of $v_1, \ldots, v_k$. Otherwise it is undefined.
$I(\cdot)$	-	$I()$ always returns v which the value taken from input tape INPUT() which is expected to contain (INSERT, v).

Table 6.4: Protocol $\Pi^{\text{alt}}_{\text{BA}}$

Lemma 11 (Agreement)**.** *Under the Honest Majority Assumption, it holds that Π^{alt}_{BA} defined in Table 6.4 parameterized with $k = \lceil 2f\lambda \rceil$ running for a total number of rounds $L \geq 4k/f$, satisfies Agreement with probability at least $1 - e^{-\Omega(\epsilon^2 f\lambda)}$.*

Proof. As stated in the protocol description, each party is constantly trying to include its input value into the chain and this means chains no longer have to contain unique values. Since the reading function $(\cdot)$ returns the majority bit of the first k blocks, in order to provide Agreement it is enough with assuring the same first k blocks of each chain are equal. By the chain growth property $\tau L = 4(1-\epsilon)k > 2k$, since $\epsilon < \frac{1}{3}$. A disagreement in the first k blocks of two given chains $\mathcal{C}_1, \mathcal{C}_2$ implies $\mathcal{C}_1^{\lceil k} \not\preceq \mathcal{C}_2$ and $\mathcal{C}_2^{\lceil k} \not\preceq \mathcal{C}_1$ which contradicts the common prefix property. □

Just as before, note that the inequality can be further fit by taking $L \geq 1.5k/f$.

Lemma 12 (Validity)**.** *Under the Honest Majority Assumption strengthened so that $\frac{t}{n-t} \leq \frac{1-\delta}{2}$, it holds that Π^{alt}_{BA} defined in Table 6.4 parameterized with $k = \lceil 2f\lambda \rceil$ running for a total number of rounds $L \geq 4k/f$, satisfies Agreement with probability at least $1 - e^{-\Omega(\epsilon^2 f\lambda)}$.*

Proof. It is enough to prove that the majority of the bits of the first k blocks of any chain contain a value present in one of the honest parties input tape. This can be proved showing that most of the first k blocks were mined by honest parties. Assuming a typical execution, since $k > 2f\lambda$, Theorem 4 can be applied and thee rate of honest blocks in any set of k(or more) consecutive blocks contains at least $\mu > \frac{1}{2} + \frac{1-\delta}{2} - (1+\frac{\delta}{2}) \cdot \frac{t}{n-t} > \frac{1}{2} + \frac{t}{n-t} - (1+\frac{\delta}{2}) \cdot \frac{t}{n-t} > \frac{1}{2} + \frac{t}{n-t} - \frac{t}{n-t} = \frac{1}{2}$ honest blocks. □

It is easy to see that the strengthened Honest Majority Assumption implies that the adversary's hashing power is bounded by $1/3$. Thus, the following theorem has been proven:

Theorem 7. *Under the Honest Majority Assumption strengthened so that* $\frac{t}{n-t} \leq \frac{1-\delta}{2}$*, it holds that* Π_{BA}^{alt} *defined in Table 6.4 parameterized with* $k = \lceil 2f\lambda \rceil$ *running for a total number of rounds* $L \geq 4k/f$*, solves the Byzantine Generals problem with probability at least* $1 - e^{-\Omega(\epsilon^2 f\lambda)}$.

Chapter 7

Conclusion

7.1 General Conclusions

A simplified version of the model described in [Can01] has been developed. This model is complex enough to support the description of the Bitcoin backbone protocol bounding resources and abstracting the communication layer while providing enough freedom to model other protocols. For instance if some of the condition of the communication layer or any of the random oracles of functionalities is changed the model can be adapted ans the protocol can be newly built on top of it.

The guts of the Bitcoin protocol have been described and analyzed in a rigorous way. The Bitcoin protocol is parameterized too which in turn means that the protocol can be used as base to build applications on by instantiating the protocol parameters. The results obtained are very powerful and the assumptions needed to obtain them are not very strong which suggests the solutions built using the protocol may be very useful in a real world scenario.

The problems solved are interesting in their own way. The problem of Maintaining a Public Ledger is very interesting because it allows the creation of transaction based applications. This protocol is parameterized too which makes it flexible allowing to instantiate the Bitcoin protocol as a particular case. The General Byzantines problem is also solved but the solutions given are not parameterized. Still they are interesting. The first one shows that Nakamoto didn't get it completely wright in his solution to the problem since *validity* cannot be assured. Meanwhile the second one manages to achieved the deterministic bound $t < 1/3$. Another solution exists based on the Nakamoto solution to the Public Ledger problem. This solution is particularly interesting because it actually improves the bound of the deterministic solution to $t < 1/2$ which clear puts in perspective the power of the Blockchain technology. This solution is not shown because there was not enough time to write it down.

All in all the work is complete and self contained and can be anyone with mathematical or computer science education which sets it apart from the papers and works used as a reference and makes it useful as an introduction to the Blockchain technology.

7.2 Objectives Analysis

Most of the objectives stated in Chapter 1 have been accomplished:

Objective	Level of Accomplishment	Comments
Studying current state of the art works	80%	The final conception of the current state of the art is almost fully complete since, although not all texts have been read or studied their existence and contents have been noted.
Analyzing the most relevant current works on Blockchain topic from different standpoints	70%	Many articles related to the properties of other blockchain protocols have not being studied in detail.
Exploring the accepted definitions of the Blockchain concept and the consequences of such definitions in terms of models and protocols	95%	This objective has been almost fully completed. A deep understanding of the Blockchain concept and its consequences has been acquired as well as the protocols linked to them. Not all computational models are fully understood.
Searching for problems that can be solved using Blockchain properties	75%	In the current work two problems are solved using the described protocol. There surely are many more but there was not enough time to research more of them.
Defining in detail the computational model upon the protocol will be based	95%	Except some very specific details the computational model has been fully understood(although based on another model it is defined by the author which denotes enough knowledge to distinguish what is necessary and what not and to what extent.)
Describing the protocol that will use the Blockchain technology and will be used to solve the stated problems	100%	The protocol is perfectly described and much care has been put in making such description as comprehensive as possible.

Analyzing the vulnerabilities of the described protocol	50%	A more detailed analysis could have been made. For instance, the perpetrated security analysis is only based on the proved properties of the protocol. A specific study on possible attacks and their consequences is needed although there was not enough time to make it.
Detailing the problems that are going to be solved	100%	Solved problems are formally defined.
Illustrating the solutions given to the problems	80%	Solutions are defined at high level even though no important details are missed. An interesting solution for the General Byzantines problem is missing.
Implementing the protocol in a general purpose language	0%	There was not enough time to do any implementation.

7.3 Future Work

A similar analysis has to be made for the main Blockchain based protocols in order to provide a clear comparison of their characteristics and capabilities. Some of the most important of these protocols differ on the Bitcoin backbone protocol mainly in the PoW, like Ouroboros Proof of Stake or Chia Proof of Space. Some work has been done in this regard in [Ren19] and [KQR20]. Also a deep analysis of the vulnerabilities of these protocols has to be made, some progress has been done in [Dem+20].

On the other hand there are many ideas on new areas where this technology can be applied, some of them are very obvious and maybe the reason for Blockchain's existence but some others are new and disruptive like using it in the healthcare sector, virtual voting, supply chains or the creation of smart contracts. Although some of these ideas are good some of them are not. There needs to be a deep understanding of the tool in hand in order to know if it is the best choice for the task. Also, once a Blockchain based protocol has been regarded the best solution for the task(like in the case of smart contracts at the current time) there needs to be a lot of development done for these ideas to see the light.

Last, the need to educate non-expert users on the matter is urgent. Blockchain being such a powerful tool in the elimination of central entities its success is of human interest. In order for this to happen people need to be educated on the Blockchain concept so they understand what it truly is and the potential it actually has over some of their particular uses. As stated in the introduction there is a lot of misunderstandings around the concept of Blockchain. Even in some advanced readings, the terms are confused and mixed. It is the belief of the author that, in order to understand the Blockchain, it has to be fully disconnected from the Bitcoin network and he hopes it this is the case in the near future. It is also believed that this technology has come to stay and it will find its place in one way or

another, maybe in a totally unexpected place.

Bibliography

[Can01] Ran Canetti. “Universally composable security: A new paradigm for cryptographic protocols”. In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE. 2001, pp. 136–145.

[Dem+20] Amir Dembo et al. “Everything is a race and nakamoto always wins”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 859–878.

[Eln18] Saeed Elnaj. “The Problems with Bitcoin and the Future of Blockchain”. In: *Forbes* (2018).

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. “Impossibility of distributed consensus with one faulty process”. In: *Journal of the ACM (JACM)* 32.2 (1985), pp. 374–382.

[Gar05] Vijay K Garg. *Concurrent and distributed computing in Java*. John Wiley & Sons, 2005.

[Gat+18] Valentina Gatteschi et al. “To blockchain or not to blockchain: That is the question”. In: *It Professional* 20.2 (2018), pp. 62–74.

[GG20] Federico Garcia and Salzer Gernot. “A Document Class and a Package for Handling Multi-File Projects”. In: (2020), p. 15.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. “The bitcoin backbone protocol: Analysis and applications”. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2015, pp. 281–310.

[Gra20] Vincent Gramoli. “From blockchain consensus back to Byzantine consensus”. In: *Future Generation Computer Systems* 107 (2020), pp. 760–769.

[GS20] JOHN M. GRIFFIN and AMIN SHAMS. “Is Bitcoin Really Untethered?” In: *The Journal of Finance* 75.4 (2020), pp. 1913–1964. DOI: https://doi.org/10.1111/jofi.12903. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jofi.12903. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jofi.12903.

[KQR20] Aggelos Kiayias, Saad Quader, and Alexander Russel. *Consistency of proof-of-stake blockchains with concurrent honest slot leaders*. https://eprint.iacr.org/2020/041.pdf. 2020.

[Mor08] Lapo F. Mori. "Writing a thesis with LATEX". In: (2008), p. 39.

[Nak] Satoshi Nakamoto. *Nakomoto General Bizantines solution*. https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html.

[Nak08] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: *Decentralized Business Review* (2008), p. 21260.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults". In: *Journal of the ACM (JACM)* 27.2 (1980), pp. 228–234.

[QL15] Genmo Qi and Peter Lu. "Consensus Protocols 101". In: (2015).

[Rad18] Nicole Radziwill. "Blockchain revolution: How the technology behind Bitcoin is changing money, business, and the world". In: *The Quality Management Journal* 25.1 (2018), pp. 64–65.

[Ren19] Ling Ren. "Analysis of nakamoto consensus". In: *Cryptology ePrint Archive* (2019).

[Rus92] John Rushby. *Formal verification of an Oral Messages algorithm for interactive consistency*. Tech. rep. 1992.

[San06] Nicola Santoro. *Design and analysis of distributed algorithms*. John Wiley & Sons, 2006.

[SKN19] Sachin Shetty, Charles A Kamhoua, and Laurent L Njilla. *Blockchain for distributed systems security*. John Wiley & Sons, 2019.

[Wea18] Nicholas Weaver. "Risks of cryptocurrencies". In: *Communications of the ACM* 61.6 (2018), pp. 20–24.

[Zha13] Zuyu Zhang. "The Oral Message algorithm for the Byzantine Generals Problem". In: (2013).