Leaning on **I**mpossible-to-**P**arallelise **W**ork for **I**mmutability **G**uarantees in the Blockchain

MSc Thesis *(Afstudeerscriptie)*

written by

Esteban Landerreche
(born 4 February, 1991 in Mexico City, Mexico)

under the supervision of **Dr.ir. Marc Stevens** and **Dr. Christian Schaffner**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
28 September 2017	Prof. Dr. Ronald de Wolf (Chair)
	Dr.ir. Marc Stevens
	Dr. Christian Schaffner
	Dr. Maarten Everts
	Malvin Gattinger, MSc

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Para mis padres, Esteban y Rossana,
porque decidieron que a mis veinte años
ya era hora de aprender a andar en bici.
No podría haber hecho esto sin ustedes

Acknowledgements

First of all, I would like to thank my supervisors: Marc and Christian. I want to thank Marc for entrusting me with this project, hoping that I have repaid that trust. I started to work on this thesis shortly before the news of the demise of SHA-1 which meant that Marc was very busy controlling the panic that ensued. Even then, he answered my questions and pointed me in the right direction. When things had cooled down a bit, he always responded with enthusiasm and encouragement and I thank him for that. He also made sure that I would have a place inside of the Crypto group at CWI and helped immerse me in the research life. When I approached Christian a bit less than a year ago about doing my thesis in cryptography, he told me that he had transcended the classical world and only had quantum topics to give me. However, he never stopped looking for someone outside the quantum realm that would have a topic for me. Fortunately, his search succeeded. While my thesis did not fall into his area of expertise, Christian was always willing to help me and answer my questions. I am very thankful for his help, even if he did not appreciate my lyrical vocabulary. I would also want to thank my committee who had interesting questions and insightful comments that made this thesis better.

I want to especially thank my parents, as I could not have done this without them. I am not talking about the fact that they were always there for me or how they supported me in every way so I could be here. They did all that, and much more, but if it wasn't for a long weekend six years ago in Valle de Bravo with them, I would not have been able to survive Amsterdam. After twenty years, they decided it was enough and that I had to learn how to ride a bike. I was not entirely convinced (as I had no idea how much that would affect my life four years later) but my parents did not let me give up and I had to learn how to ride a bike. Looking back, learning how to ride a bike also taught me that it is always worth it to give that extra bit of effort to conquer the things that challenge me the most, as the rewards will be worth it. I want to thank my mom for pushing me to come here even if it meant being across the ocean, something I know she does not appreciate. My father, on the other hand, pretends that he is glad that I am not there to hug him. I want to thank him for pushing me to always better myself and work harder. I want to dedicate this thesis to them, for always pushing me to reach my goals, even if it means that I will be far away from them, and for always being there with their love and encouragement.

Not content with giving me everything I have already mentioned, my parents also gave me two wonderful sisters. I know that my sister Paula is very happy that this program pushed me farther away from the philosophical view that is prevalent at the ILLC. I thank her for many fun, intense talks that we had and I regret all the ones that we could not have because of the distance. I am yet to find someone who works as hard as my sister Maite, who always worries about not only bettering herself but also about making everyone around her better too. I want to thank her for her kindness and let her know that she will achieve anything she puts her mind to. I also want to thank both my sisters for that week in Madrid that helped me restore my sanity in times of thesis writing. I am very thankful and proud to call both of them my sisters and I look forward to eventually solving the world/writing a paper with them as a three-headed sibling dragon.

These two years in Amsterdam have been defined by multiple people. Starting from my mentor and co-author David Fernández-Duque. He introduced me to mathematical research on purpose and to the ILLC by mistake. It is not a stretch to say that without him I would never have been here. My arrival in Amsterdam was complicated to say the least and I want to thank Michael and Stella for preventing me from being homeless when I got here (not that it stopped people from believing that was the case). I am very grateful to Pablo, Chris and Sirin for sharing their home with me and giving me a home. I will always appreciate my time in the Bijlmer and all the commander games that took place in there. Speaking of homes, I would also like to thank Marianne for opening the doors of hers for me and worrying about me .

My experience in the ILLC couldn't have been as good without the help of Tanja, Fenneke and the rest of the ILLC office. I want to thank my mentor Floris, who lent me an ear when I needed it. I also need to thank many of the lecturers who taught me many things. I want to especially thank Julian Kiverstein, who made me rethink the way I think about thinking. I also want to thank Jeroen and Malvin, for having the patience to help me when they were TAs and I either pestered them with questions or sent my assignments just at the deadline.

At the ILLC I met many important people that made my life better. There is Lucy, who was always up for a nice discussion and also put a lot of effort into making life a lot better for all logicians through Ex Falso. Many of my fondest memories in Amsterdam include Pablo and the good times we had together, including general geeking out and football discussions. I also want to thank him for the effort he put in our RPGs, where I had good times with him, Chris, Julian, Dan and Fernando. Also a strict practicant of the art of not saying much, Jakob was always there to lend a helping hand and a good time (Freiburg is a nice place that I had the pleasure of visiting with him). Jonni was always there to bring a smile and a bit of common sense, as well as a lot of patience to deal with my bad game theory jokes. I had some very nice conversations with Joannes, which he apparently did not enjoy as much as I did because we agreed on too many things. I regret never going to a metal concert with Anthi, and in general not sharing enough music (as her taste in music seems to be critically declining). I will always remember people thinking that Melina and I were fighting when we relived the eternal struggle between capital-dwelling snobs and, as we call them, provincians. Many other people colored my time at the ILLC, too many of them to mention (I tried, it was too much) and I want to thank all of them. I also want to thank the Crypto group at the CWI who accepted me as one of their own and always had interesting comments about my shirts. Especially my office mate Mark who proofread this thesis and was always there with a useful comment or interesting conversation.

I also want to thank all my friends back home, especially the ones that have to endure my never-ending "*Sorry, I can't make it.*" attempts at jokes. I especially wish to thank Gabriel, Lorenza, Mariana, Gunther and Emmanuel for coming all the way here to see me. Felipe and Diego get a little less thanks than the rest because they had a chance to visit me but didn't. I also want to thank everybody back home who has been rooting for me these last two years.

Finally, I want to thank Jana for everything we have lived together these last two years. In the process of writing this thesis, she was the one that had to suffer my stress and frustration. This requires a lot of patience and some work, but she managed admirably. She was always there to give me cuddles when appropriate but also to tell me when I was being just hard on myself and make me stop it. She explicitly asked me to thank her for reading and proofreading my thesis, so I will avoid doing it. However, the real reason why I am thankful for Jana is because she has made these last two years just so much better. Her smile and earnest excitement about life remind me to enjoy the nice things in my day to day. She is also stubborn, always holding a mirror up to me and not letting me escape so I can grow to be a better person. She is intelligent, beautiful and (most importantly) she laughs at my jokes. This Masters would not have been what it is without her and for that I am grateful.

Abstract

Blockchains are structures that allow to establish trust by relying on cryptographic primitives to ensure that the information encoded in them cannot be changed. Bitcoin is the first example of a blockchain and an important amount of the research is concerned with replicating its advantages in other settings. Another avenue of research focuses on improving on the flaws of Bitcoin, like how it incentivises parallelisation and is vulnerable to quantum attacks. An important limitation of Bitcoin is that its immutability guarantees can only be maintained in a large network at a large cost, making it unusable for many applications. In this thesis, we present a blockchain protocol that avoids these issues by ensuring immutability through proofs of work based on *sequential computation*. By separating the proofs of work from the consensus mechanism, we avoid the incentives for parallelisation found in Bitcoin while maintaining similar guarantees that the information contained within cannot be changed. First, we present the security guarantees that serial proofs of work contribute to the blockchain structure. We then construct a protocol in a modular way through the universal composability framework in an idealised setting and prove that it is secure. Next, we get rid of many of the idealising assumptions and show that our model is still secure. Finally, we introduce a new setting for the use of blockchains, with peers maintaining personal blockchains that form a web of trust. We believe that the models presented in this work will be able to replicate the immutability guarantees of Bitcoin in a permissioned setting while avoiding some of the setbacks of that model.

Contents

List of Symbols and Abbreviations

$\mathcal{Z}$	Environment
$\mathcal{A}$	Adversary
$\mathcal{V}$	Verifier
$\mathcal{P}$	Prover
i	Round
a_j	Participant with index j
pk_j, sk_j	Public and secret keys of participant a_j
n	Total number of participants
Q	Percentage of honest parties
i	Number of round
λ	Security parameter
τ	Round time parameter
ω	Minimum strength function with parameters ω^* and t^*
γ_j	Rate of participant a_j
$\gamma_{\mathcal{A}}$	Rate of the adversary
γ^*	Practically achievable maximum rate
PV game	Prover-Verifier game

BC	The ledger chain
$BC[i]$	The i-th block of the ledger chain
NA	The set of participants active when the block was made
st	The round the block was created
$link$	The hash of the previous block in the chain
T	The set of transactions
G	Proofs of cheat
W	Set of proofs of work
Sig	Set of signatures of the block

BC^j		The PoW chain of a_j
$PC^j[i]$		The i-th block of a_j PoW chain
	id	The index of a_j
	st	The round the block was created
	$link$	The hash of the previous block in the chain
	$linkLedger$	The hash of the last block in the ledger chain
	G	Generic information
	W	Proof of work over $link$
	Sig	a_j's signature of the block

BFT	Practical Byzantine-Fault Tolerance protocol
Consensus	Protocol to choose and construct a block
SingleLipwig$^\tau$	One PoW chain model
SingleVarLipwig$^\omega$	One PoW chain model with proofs of work of variable strength
IdealLipwig$^\tau$	Idealized model ledger protocol
Lipwig$^\omega$	Ledger protocol with proofs of work of variable strength
Semaphore	End-of-round beacon for Lipwig$^\omega$
TxsPool	A pool of transactions maintained by each participant
verifyBC	Verification function for ledger blocks
verifyTxs	Verification function for transactions
verifyPC	Verification function for PoW blocks
verifyPoW	Verification function for proofs of work
checkCheat	Cheater discovery function
PoW	Proof of work function
H	Hash function
ind	Outputs indices of participants who contributed a certain element
str	Strength of a proof of work
len	Length of a blockchain

1 Introduction

Trust is one of the fundamental building blocks of our society. At the same time, building trust is a long and difficult process, as humans are selfish and fickle. As trust is a necessary component in any interaction between individuals, institutions that act as facilitators of trust have been an essential part of civilisation. Two people that do not trust each other can make use of an intermediary that they both trust to conduct any operation that requires confidence. The need for a facilitator of trust is considered an unfortunate necessity, as it involves additional costs that one would prefer to avoid, but is still easier than building trust with every other person. An important issue with this system is that it still requires to put trust in people, either directly or indirectly, which opens the door to the possibility of corruption. Ideally, there could be a way to transfer that trust into something that cannot be compromised, something that lies beyond the control of anyone. Mathematics is something that complies with these characteristics, but can we use it to solve this problem?

Enter Bitcoin. In the present world, the value of a currency is linked to the government which issues it. However, Bitcoin has no institution backing it, because it does not need one. Bitcoin is a transaction ledger built over a blockchain, a data structure which, as its name suggests, consists of a chain of blocks of information. The advantage of this structure is that any block can only be changed if every block that comes after it in the chain is changed as well. This, combined with the fact that creating a block requires serious computational investment, means that no one can (realistically) change what is written in the blockchain. This is made possible by *proof-of-work* functions. Originally created as a way to defend against spam email, proofs of work are cryptographic puzzles that require applying a hash function to partially random inputs until the output has a certain property. The choice of hash functions ensures that there is no option other than brute force when trying to solve a proof of work. Therefore, changing a record requires a lot of work and time. In practice, it becomes virtually impossible to change any record in the past. This means that anyone has the assurance that whenever a transaction is recorded it cannot be erased. Similarly, transactions cannot be back dated to appear to have happened before they did. The cryptographic assurances permit anyone to trust the blockchain, without needing to trust any of the people maintaining it.

Bitcoin achieved something that was long thought to be impossible. It created a currency that people trust without the support of a central bank or government. This has created a lot of buzz around the idea of blockchains, with blockchain-based solutions appearing for anything from property registries to healthcare records. The ability to bypass intermediaries for cheaper and more efficient systems can affect any aspect of modern life. However, not all problems are created equal and blockchains are not a one-size-fits-all solution. The Bitcoin blockchain can only realise its full potential in a particular setting. Additionally, it has many problems, including serious scalability and sustainability concerns. The purpose of this thesis is to build a blockchain that avoids the issues of Bitcoin without sacrificing the property of *immutability*. Our system does not intend to change or substitute the Bitcoin blockchain, but presents an alternative that can maintain the same guarantees in a different setting.

1.1 Contributions of this thesis

The main contribution of this thesis is the presentation of an alternative proof of work for blockchains. We seek to address the issues inherent to the Bitcoin blockchain structure, particularly the fact that its security is directly linked to the amount of processors computing the proofs of work. This makes it unfeasible for this blockchain to be used efficiently in smaller networks. Additionally, it implies a tremendous waste of energy in order to maintain the correct functioning of the network. With a lot of interest in using blockchains in private networks (which are considerably smaller), we present a blockchain that can conserve similar immutability guarantees even when the chain is maintained by only one agent. Our security guarantees are similarly based on computing power, but in such a way that participants have no incentive to parallelise the work. By ensuring that the work to be executed must be sequential, we are only interested in the computational power of a single core. This makes our security assumptions stronger, as they are based on the speed of individual processors. We achieve this by ensuring that our proofs of work are *serial*: every step takes the output of the previous step as an input.

We present notions of security for blockchains based on clock time, where we consider a blockchain secure based on how much time it takes to build a different blockchain that is structurally indistinguishable from it. We also show that our proofs of work act as a timestamping mechanism, proving that certain records could not have been added after a certain point in the past. These two notions are similar to the properties found in Bitcoin that we wish to emulate, although in a very different setting. We will focus on a *permissioned* setting, where all participants are known and not anyone can join. Here we will build an idealised blockchain protocol using our proof-of-work chains and we will prove that the resulting blockchains are secure. Our protocol will take advantage of the setting to do things that cannot be done in the Bitcoin blockchain, like the elimination of misbehaving agents. After this, we will present a more realistic protocol, prove its security and enhance it with a randomness generator. We will follow this by presenting a new setting for blockchains, where they can be used and maintained by individuals creating a web of trust.

In Terry Pratchett's book *Making Money*, Moist von Lipwig is put in charge of the Royal Mint and Bank and tasked with making a convoluted system work again by disrupting the current status quo with new ideas. Similar to the way Bitcoin achieved the impossible by shifting trust from a central bank towards an incorruptible force, cryptography, Moist von Lipwig shifts the value of currency from gold towards the work of golems. Because they are both incorruptible, trusting in either cryptography or golems permits the system to work, even when people are purposefully trying to stop this. Our protocol Lipwig takes inspiration from this fact, as we will **l**ean on **i**mpossible-to-**p**arallelise **w**ork to achieve **i**mmutability **g**uarantees. The golems that safeguard our protocol will be our serial proofs of work, incorruptible and always at work.

The structure of this thesis is as follows: First, in Chapter 2 we will briefly present the history, challenges and advantages of blockchain. We will also present the serial proof-of-work functions which will support our protocol as well as a base for universal composability, the modelling paradigm we will use in this thesis. In Chapter 3, we will present the concept of proof-of-work blockchains or PoW chains and prove that they provide the immutability guarantees that we want. In Chapter 4 we will define the necessary components to build the model and we will present an idealised version $\mathsf{IdealLipwig}^{\tau}$ with a set round time τ. Chapter 5 will relax some conditions of the previous model and construct Lipwig^{ω}, a less rigid model where proofs of work may vary in strength (but must be at least as strong as ω). Finally, in Chapter 6, we will abandon the classical blockchain setting and present a model of independent, personal chains which are secured through a distributed web of trust.

Preliminaries 2

In this chapter we will present the necessary background for the thesis. First, we will present a survey of blockchain literature, focusing on academic work but touching on the most important points of real-world implementations. We will then present the theoretical background behind our serial proofs of work, which are the essential building blocks of our construction. Finally, we will present the universal composability framework which we will use to construct the protocols in Chapters 4 and 5.

2.1 The Blockchain

A classical problem in distributed computing is that of consensus. Is it possible to have different agents agree even when some of them are actively trying to prevent it? This problem becomes relevant in the context of computer science because processors can, and will, fail and the computation must still succeed. This problem is known as the Byzantine Generals problem (or in a more practical context: Byzantine Fault Tolerance) and was originally presented in [PSL80] under another name. This setting considers participants that can act arbitrarily and/or maliciously. Numerous solutions for this problem have been found [CL$^+$99, LVCQ16, MXC$^+$16], under different network properties and with varying communication complexity. Original solutions of this problem all considered a setting in which all participants are known from the beginning. In [Oku05] it was shown that if the network is unknown, it is impossible to create agreement over all the parties, even if there is only a single participant which does not follow the protocol. This problem arises from an attack commonly known as Sybil, in which the adversary creates multiple identities to overrun the network. Without a way to prevent a participant from arbitrarily creating new identities, it is impossible to have a way for the system to work. The possibility of reaching consensus in an unknown network became especially relevant after the rise of the internet. However, no practical solutions appeared until the advent of Bitcoin.

2.1.1 Bitcoin

After the 2008 financial crisis, trust in financial institutions was gravely shaken. That year, someone (or someones) calling themselves Satoshi Nakamoto presented a system called Bitcoin that would remove the need for a central bank to maintain a currency [Nak08]. In 2009, the network implementing this system came into existence. Bitcoin is essentially a ledger maintained by a network of participants called *miners* which maintain the ledger through the internet. Any person can become a miner simply by installing the code and running the protocol. Consensus is achieved in this network where anyone can join through the existence of proofs of work. Originally presented as a way to prevent spam in [DN92], proofs of work are cryptographic puzzles where someone runs a hash function over certain inputs until the output fulfills a certain property. The addition of these proofs of work prevented the possibility of a Sybil attack, as any participant's power in the network is determined by the amount of computational power they have access to, not by the number of identities they hold. Bitcoin promised to create a decentralised, democratic and self-sustaining currency independent of any individual entity's control and (mostly) fulfilled those promises.

Bitcoin introduced the structure of a blockchain which consists of a series of blocks chained together through hash pointers. Each of these blocks contains a part of Bitcoin's ledger, which grows as new blocks are added to the chain. To add the next block of the chain, any miner can take the transactions that exist in the system, order them and create a block containing them and a pointer to the last block in the chain. After that, they must repeatedly input the candidate block and a random nonce to the SHA256 function until the output has an initial segment of a certain length that consists of only zeroes. If they manage to do this, they send the block and the nonce to all the other participants in the network. These participants can check whether the desired property is met and therefore accept the block. After this, every miner creates a new candidate block pointing to the newly mined block. Miners will only accept blocks that fulfill this property, making it impossible for anyone to arbitrarily create a chain, regardless of how many participants they control. This system is exactly what allows Bitcoin to survive Sybil attacks, as it is irrelevant how many identities a person has. The only way a participant can gain more power in the network is by acquiring more computational power.

Bitcoin is adaptive and responds to the amount of power that is being invested in it. The system expects a new block to be created every ten minutes of real time and will update the difficulty of the proof of work accordingly to maintain this time between blocks. A reason for this wait time is the possibility that more than one miner finds a valid block at similar times. When a miner gets a chain that differs with their own, they keep the longest one and work over that one. If both chains are the same length, the miner continues to work over the one they currently hold[1]. This introduces some practical issues, as something might disappear from the blockchain as a different, longer, chain appears. However, after a block has a certain number of blocks that follow it, blockchains that do not contain that block will be more and more rare, up to the point where that block is considered a permanent part of the chain[2]. The blockchain structure then prevents anyone from changing older blocks.

The proofs of work not only serve to choose which participant has the right to create a new block, they are also fundamental in ensuring that the ledger cannot be changed. If someone were to change any record in a block, the hash of the new block and the nonce would no longer fulfill the necessary properties to count as a valid block (with practical certainty). Therefore, a new nonce must be found for the new block to be accepted by the rest of the miners. While this would take some time, it is not enough to ensure that the ledger cannot be changed. Immutability is achieved thanks to the blockchain structure. The blocks of the blockchain are connected through hash pointers (also using SHA256), which means that changing a block implies changing the next one, as it needs to update the hash pointer to the previous one. Therefore, changing a block implies changing all of the ones that came after it, including the ones that are being added to the chain while the process of rewriting the chain is going on. Supposing that the party trying to change the blockchain does not control more than half of the computational power, their modified chain, or *fork*, will grow slower than the chain. Therefore, the fork will never be long enough to substitute the chain. This implies that once something is found on the chain with enough blocks in front of it (deep enough in the chain), it will be there forever.

All of this machinery is needed to realise the goal of Bitcoin: creating a currency that is not controlled by a central authority. The ledger registers all the transactions that are done with Bitcoins. To prove that someone has enough money to do a transaction, they can show the records of receiving the money. The receiver can also check whether the money has been already spent or not. For the system to work, it must be impossible for someone to spend the same money twice. As long as they control the money, someone could create as many transactions as they want, but only one will be added to the chain. Therefore, this must happen before the transaction can be considered complete. After a transaction is encoded in a block that is deep enough, the transaction may be considered as final. As we saw before, it becomes almost impossible for anyone to erase this transaction so they can spend the money again. This system does bring up some questions: if Bitcoin is just a ledger of transactions, where does the money come from? New Bitcoins are minted every time a block is created, and they belong to the party mining the block. This is the way someone may

[1]The Bitcoin system no longer works like this, now the one with the highest difficulty is chosen, but this technicality is not important for this presentation.

[2]In practice, a transaction in Bitcoin should not be considered completely finalised until the block it is found on has five blocks after it.

acquire new Bitcoin, but it also solves a separate problem. Mining new blocks is a time and energy consuming process, so the miners should have an incentive to do it and maintain the network running. By rewarding miners for creating new blocks, the system maintains itself.

The great triumph of Bitcoin is that it was able to engender trust in a setting where there was none. Bitcoin is maintained by a network of parties that do not know each other or their motivations and have no reason to trust each other. The system works because parties transfer the trust they have in the cryptography unto the other participants. Bitcoin allows mutually distrustful parties to trust each other based only on the cryptographic security offered by the blockchain. Before the appearance of Bitcoin, reaching consensus in an unknown network was thought to be impossible [Oku05].

While the appearance of Bitcoin was sudden, its parts come from at least thirty years of research in computer science. The blockchain structure comes from an attempt to create a time-stamping mechanism for digital files [BHS93]. Here, documents are ordered relative to each other by forming a chain, where each document points at its predecessor and is signed by its creator. Therefore, if someone gets a document from a trusted source, all the documents that precede that document can be considered ordered. Our blockchain will actually realise this goal with the added advantage that no trusted party is needed, as the serial proofs of work will take its place. Merkle trees are another important component of Bitcoin, as they permit an efficient way to store and verify information [Mer80]. Not even the idea of using proofs of work to create something akin to *electronic cash* is new to Bitcoin, as a system known as *Hash cash* was created in 1997 [Bac01], which used hash-based proofs of work as cash. However, it relied on a central authority and had no built in mechanisms to protect from double spending. The true contribution of Bitcoin is taking all of these disjoint pieces and putting them together in a real-world system [NC17].

Due to the fact that Bitcoin appeared seemingly out of nowhere, it took some time to formally understand how it worked. The first comprehensive paper in Bitcoin presented the abstraction of the Bitcoin blockchain and proved its security in a partially synchronous setting [GKL14]. This paper was followed by numerous other papers presenting different aspects of Bitcoin. The same team followed up their work with a proof of Bitcoin with chains of variable difficulty in [GKL17]. In [PSas16], Bitcoin is proved secure in the asynchronous model and [BMTZ17] presents a fully-composable treatment of Bitcoin. Other work has shown ways in which the Bitcoin blockchain can (or cannot) be used, like [BCD$^+$14] which presents the possibility of sidechains that depend on the Bitcoin blockchain and [PW16] which shows the issues with using Bitcoin as a random number generator. Academic efforts have also helped to find issues with the Bitcoin model, most notably [ES14b] showed that Bitcoin miners are incentivised to deviate from the protocol in a certain way in order to maximise their profit In other words: Bitcoin is not incentive compatible.

Unfortunately, the fact that it is not incentive compatible is not the only problem facing Bitcoin. In practice, Bitcoin has failed to achieve some of its goals. For starters, the implementation of the puzzle made it advantageous for miners to join in groups, known as mining pools, in order to minimise the variance of the payouts [MKKS15]. This means that the mining power was concentrated in groups, instead of being completely distributed over the miners. This meant that the possibility of a certain entity having more than half of the mining power became a realistic possibility, something that seemed unfeasible from a purely theoretical standpoint. This happened in June 2014 when a mining pool known as GHash controlled 51% of the mining power [ES14a]. Fortunately, this situation was resolved without the network being affected. The issue of centralisation was further compromised when it became profitable for investors to build Bitcoin *mining farms* to acquire rewards. Bitcoin mining became a feasible business operation where maximising profits goes hand in hand with scaling towards large operations, which further affected individual miners and centralised the network. This reality is made possible by the possibility of *parallelising* the computations necessary to compute proofs of work. An unintended consequence of this is the energy that it takes to maintain the network due to the difficulty increase caused by having specialised mining facilities. It is estimated by [ene17] that the Bitcoin network uses almost the same amount of electricity as the whole country of Iceland. This has direct economic and environmental consequences which make the current system unsustainable in the long run. Looking towards the future, it has been shown that quantum computers need less work to find a valid proof of work per block due to an algorithm known as Grover's search [Gro97]. This means that someone with access to a quantum computer has an advantage when issuing blocks and could possibly gain control of the network. Additionally, quantum computers could be used to undermine the immutability of the blockchain, as it could become feasible to fork the blockchain. There are other, more practical, issues with the Bitcoin blockchain, over which there are very contentious arguments. These issues, however, are not of particular interest for us in this setting.

2.1.2 Beyond Bitcoin

While Bitcoin was the first implementation of a blockchain, it is far from the only one. In 2015, Ethereum appeared [Eth16]. Based on the proof-of-work paradigm presented in Bitcoin (also known as the Nakamoto paradigm), Ethereum extended the abilities of blockchains by building a platform for *smart contracts*, that is, contracts that can execute on their own. These contracts are maintained in the blockchain and remove the need of an intermediary to guarantee the fulfillment of a contract. This system has multiple applications that go beyond a simple cryptocurrency and it has created even more interest in the applicability of blockchains. There exist many other blockchains with different implementation goals. Both zcash [Wil16] and Monero [vS14] try to guarantee privacy, using zero-knowledge proofs and ring signatures respectively.

The issue of sustainability of the blockchain is something that must be solved for blockchains to become widely used in other applications. The search for alternatives to Nakamoto for a consensus protocol for permissionless networks has been the focus of considerable work. A setting known as proof of stake, where the creator of the next block is chosen by a lottery where the odds correspond to the amount of money they control, has been suggested and widely studied. While it was originally questioned because of the possibility of an attack known as *nothing-at stake* [Poe14], there have been numerous proposals for this system. In [KRDO17], a provably secure protocol is shown, based on a multi-party-computation implementation of coin-flipping which requires a highly synchronous network. In [DPS16], they present a system that is robust against participants that routinely disconnect from the network. The protocol in [Mic16] presents a hybrid proof-of-stake/byzantine-fault-tolerance setting based on random information encoded in the blocks. All three protocols solve the issues presented in [Poe14] in different ways. Work in proof of stake is not limited to academic efforts, as Ethereum plans to switch from a Nakamoto paradigm to proof-of-stake consensus with their own implementation of it [But17].

Research on blockchain primarily focuses on the consensus mechanism, but it is not limited to proof-of-stake. In [PS16b], byzantine-fault-tolerance is used in conjunction with proof of work to create a protocol that is responsive, that is, it depends directly on the delay of the network and not on a bound. Although presented as an improvement over Bitcoin, [KJG$^+$16] presents a different way to use byzantine-fault-tolerance in a permissionless setting. The work in [PS16c] focuses on creating a system that is robust against what they call *sleepy* participants, players who regularly disconnect from the network.

While one of the fundamental contributions of Bitcoin was the possibility of creating trust in a network where anyone can join without permission, blockchains also have a purpose in a permissioned setting. The creation of trust between mutually untrusting parties still has a place in permissioned networks. Because in a permissioned setting all the participants are known, Nakamoto consensus is not necessary and can be substituted by byzantine-fault-tolerance, which is considerably more efficient. The use of blockchains and their potential in permissioned networks have revived study in this field and new methods have been created, that are more robust [LVCQ16] and specifically tailored for use in blockchains [MXC$^+$16]. While some people have voiced concerns about these implementations [Sir17], the stronger setting permits the network to sacrifice robustness for efficiency and scalability. We will create a permissioned blockchain that solves some of the issues of existing permissioned blockchains, in particular immutability. Permissioned blockchains sacrifice the strong immutability guarantees provided by proofs of work, but we will see that this is not necessary. Currently, implementations of private blockchains depend on the Bitcoin blockchain to prove that the information has not been modified. For example, Exonum[3] adds a hash of the state of their blockchain to a Bitcoin transaction in order to have a proof of immutability. This process of adding pointers to a blockchain has been used to timestamp events [GMG15]. We will use a similar process to secure our own blockchain, although we will not depend on an external blockchain for it.

There has been some work on distributed ledgers that do not share the same blockchain structure as Bitcoin. In [PS16a], a blockchain is presented where transactions are not added directly in blocks but in *fruits* which are then contained in the blocks. A scaling proposal for Bitcoin, Bitcoin-NG [EGSVR16], proposes having two different types of blocks, one type of block that determines who is allowed to record transactions and microblocks which actually contain these transactions. Other more extreme proposals include the appendix of [Mic16], which presents the concept of *blocktrees*, which are a combination of blockchains and Merkle trees. Another proposal known as Mimblewimble claims to provide privacy in a blockchain that remains short by being able to eliminate transactions that are no longer relevant (the money has already been spent)[Pev17]. A modification of particular interest is the one found in Hyperledger's Fabric blockchain [Hyp17]. Built for a permissioned setting, participants running Fabric only save the records that they are involved with and not all of them, as is done traditionally. The modification to the structure which we will present is minor compared to some of these proposals but has far reaching consequences.

2.2 Serial Proofs of Work

Many of the scalability issues on Bitcoin are related to the proof of work. As we mentioned before, the fact that the work can be parallelised causes many issues. On the other hand, the immutability guarantees it provides are dependent on the amount of computational power invested in the network. Therefore, a small network will not enjoy the full advantage that proofs of work provide unless they artificially invest a lot of computational power, which is counterproductive. We will attempt to solve this problem by using serial proofs of work; by *serial* we mean functions where parallelisation does not provide any advantage. We want to have a function that proves that a participant invested enough time computing it. Functions like this have been used in order to time-lock information [MMV11], that is, encrypt information in such a way that it can be decrypted by anyone after an amount of time has passed. This requires it to be quick to encrypt but slow to decrypt. In our case, we will use it the other way around: slow to compute and quick to verify. Similar to the Nakamoto paradigm, the parties will apply this function to a block in the chain. We will then demand some properties for our proofs of work: unpredictability, easy verifiability and practical impossibility of precomputation.

[3]`http://exonum.com/index`

These three properties are all properties that would be desirable if we were interested in creating public and verifiable randomness. To ensure fair randomness, one could want to permit outside participants to contribute to the random seed, without giving them a way to influence the random output. In [LW15], the authors present a source of public randomness that is publicly verifiable as well as being able to be contributed to by anyone. The unpredictability of the model comes from the fact that it takes a certain amount of clock time to compute the random number. This prevents an adversary from influencing the seed in such a way that the probability distribution of the output is modified to her advantage. Although this is theoretically possible, an adversarial party would have to have access to a significantly faster processor to be able to find out how to modify the seed in such a way that the result has the expected properties. This particular fact about the function, that it takes at least time τ to compute (but considerably less to verify whether it was computed correctly), is exactly what we want for our serial proof-of-work function.

We will therefore base our serial proof-of-work function on the function *sloth* defined in [LW15]. This function is based on modular square roots and is similar to the one found in [JM13]. The advantage of using modular square roots is that the only way we know how to compute them is by squaring the input repeatedly until we arrive back to it. On the other hand, verification consists simply of squaring the root that we found. This provides the asymmetry between the computation and verification time. Currently, the fastest way to find a square root of a number is by performing $\log_2(p) - 2$ squarings over it. These operations cannot be parallelised, as the result of each squaring is necessary to perform the next one. While there is no way to ensure that this is the fastest way to execute this computation, we are comfortable making the assumption that no faster algorithm will be found. Assumptions of the sort are common in public-key cryptography, where security lies on the impossibility to speed up number-theoretic computations.

It might seem then that the way to proceed would be to find the minimum p such that computing a modular square root takes a time greater than τ given the assumptions over the rate of the participants computing it. There are two issues with this approach. The first one is the size of p, which would have to be so large as to be unwieldy. More importantly, it would provide a structure that is too rigid, as changes in the rate or the expected run time would imply a new choice of p. Therefore, it is better to find a significantly smaller p and then iterate the modular square root as many times as we need. The fact that we are iterating a function permits us to modify the runtime of the function as we require by changing the number of iterations. This flexibility allows us to adapt the proofs of work for advances in computing power or simply to change the time that it takes for blocks to be issued.

The choice of p does not depend solely on its size. If p is a prime such that

$p \equiv 3 \bmod 4$, for every $x \in \mathbb{F}_p$ we know that either x or $-x$ is a square. We also know that we can calculate the square root of x by raising it to the $\frac{p+1}{4}$-th power. Of course, every square (except 0) has two distinct square roots: y and $-y$. To determine which of the two roots we are interested in, we see that y and $-y$ have different parities when we use the canonical representation of elements in the field (so $-y = p - y$). Therefore, we are interested in computing the function that, given an x, first checks whether x is a square. If it is a quadratic residue, it outputs the even square root of x, otherwise, it outputs the odd square root of $-x$. Note that this function is a permutation over the field. The issue with repeatedly iterating this function is that there is a way to shorten the computation time through analytic means. Therefore, we must add another permutation between each iteration of the function. This permutation must be easy to compute in both directions and prevent further shortcuts. It was shown in [LW15] that a permutation that adds one to odd numbers and subtracts one from the even numbers is fulfils this purpose. However, because of the large amount of instances of the function being called, we might be interested in using a permutation that varies depending on the input, in order to stop the possibility of precomputation. This, however, is not something we take into account in the current model. In this thesis, we will call the composition of the square-root function and the permutation PoW. We will use it to instantiate a process Golem that is similar to *sloth* but can be run for any amount of time, continuing to iterate the function.

2.3 Universal Composability

The appearance of Bitcoin preceded any formal study on its properties. While [Nak08] explained the basic ideas behind how and why Bitcoin worked, a formal model of security of it did not appear for a couple of years. The main challenge for modelling was the fact that a framework for the study of blockchain structures did not exist. Instead of taking a concept and defining it according to a framework, it was necessary to find a setting which could properly express all the necessary properties of the Bitcoin protocol. In [GKL14], a version of Universal Composability (UC) [Can01] was used. Further study in blockchain has followed this modelling technique, both to study Bitcoin or to describe new protocols. This thesis will be no different, as we will use an extended version of UC presented in [CDPW07], which adds a global setup to the model. We choose this extension because we will assume the existence of a public-key infrastructure for all our messages. As our work corresponds mostly to the permissioned setting, it is natural to think that there exists a global setup over which the protocol will be built.

The primary idea behind universal composability is having a general model for security analysis that captures composable protocols instead of making individual models for each application [Can16]. One of the greatest advantages of this modelling technique is that it helps construct modular protocols, where parts can be changed without affecting the security of the whole. The basic idea of the model is simple: given an environment $\mathcal{Z}$ and a functionality φ, if there exists a protocol Π which realises φ in such a way that it is indistinguishable to $\mathcal{Z}$ whether it is running φ or Π then we say that Π securely realises φ. However, $\mathcal{Z}$ does not generally run Π directly, but does it through a simulator. This simulator is added because otherwise φ and Π would have to be equivalent in order for them to be indistinguishable for $\mathcal{Z}$. The main implication of this idea is that security of a system is reflected only in its effects on the environment, not in the actual structure of the protocol.

The UC framework works over interactive Turing machines (ITM) and contains two probabilistic polynomial-time algorithms $\mathcal{Z}$ and $\mathcal{A}$. The environment, represented by $\mathcal{Z}$, is the algorithm that is running the protocol and the one that must not be able to distinguish between an ideal functionality and a protocol that realises it. The parties that run the protocol are instantiated by $\mathcal{Z}$ and the environment gives them an initial input and sees their output. The adversary $\mathcal{A}$'s purpose is to interrupt the execution of the protocols in such a way that the environment $\mathcal{Z}$ can distinguish a protocol from an ideal functionality. While the adversary is limited by the setting, it is allowed to take over participants and deviate from the protocols being run. The adversary is also in charge of delivering the messages of each participant. While it is not allowed to drop messages, it can decide to change the order with which they are delivered and can delay them up to a certain point. If $\mathcal{A}$ is unable to affect the protocol in a meaningful way for $\mathcal{Z}$, the protocol is considered secure.

To prove whether a protocol successfully realises an ideal functionality, we must show that if the correct conditions are met, the environment $\mathcal{Z}$ will not be able to distinguish whether it is the protocol or the ideal functionality that is being run. Because the ideal functionality and the protocol are different (otherwise, it is not interesting), then the environment could very easily see the differences during the execution. However, the environment cannot directly interact with the protocol's execution. In particular, the adversary is charged with delivering messages between the parties, which the environment cannot access. What we must then prove is that given a compliant execution (that is, one which fulfills the properties we expect from it), the view of the environment is the same as the view the environment would have of the ideal functionality. Note that we are only interested in executions of the protocol that follow the properties that we have set out, in particular those that have the correct amount of participants where adversarial parties do not exceed a certain proportion. The environment can only act in ways permitted by the protocol and at the appropriate times. The adversary can arbitrarily deviate from the protocol as long as certain limitations are met. Therefore, we will be interested in pairs of $\mathcal{Z}$ and $\mathcal{A}$ that conform to certain properties for each particular protocol. The environment $\mathcal{Z}$ can create parties according to what is determined in the protocol. After they are created, they will follow the protocol and $\mathcal{Z}$ will only be able to interact with them in predefined ways. The environment cannot directly communicate with the adversary $\mathcal{A}$ either.

The UC model was created with multi-party computation in mind and therefore it generally cares about information leakage when computing the protocol. Because the honest parties are not attempting to keep anything secret, information leakage is not relevant in the blockchain setting, so we will ignore it. We are only interested in the ability of the adversary to modify the protocol so it does not do what is expected. UC can also be very fine grained, focusing on the ports of each participant and how they are connected. We will avoid this dimension of the model by assuming the participants have access to an ideal broadcast functionality that permits them to communicate. There is no need for private communication in the main blockchain abstraction (which does not have to be the case for the actual implementation). We will use a formalisation similar to the one used in [PS16b], where we do not explicitly define an ideal functionality, but only the properties that we expect from it. We then prove that the protocols we present realise these properties given an appropriate pair $(\mathcal{Z}, \mathcal{A})$.

There always exists the possibility that a participant might guess a signature or that two inputs to a random oracle have the same outputs. To ensure that this does not cause problems, these events must be so unlikely that their probability of occurrence is *negligible*. To quantify this, we must introduce the concept of a **negligible function**. We say that a function negl is negligible if for every positive polynomial poly there exists a positive integer z such that for all $x > z$ we have that $|\mathsf{negl}(x)| < {}^{1}/_{\mathsf{poly}(x)}$.

To prove that a protocol Π works as expected, we will define a random variable denoting the view of all participants in the protocol given that $\mathcal{Z}$ and $\mathcal{A}$ are probabilistic polynomial-time algorithms. The random variable $\textsc{exec}_{(\mathcal{Z},\mathcal{A})}[\Pi]$ is defined over all the random coins of all n participants, $\mathcal{A}$ and $\mathcal{Z}$ as well as the random oracles. Every instance of $\textsc{exec}_{(\mathcal{Z},\mathcal{A})}[\Pi]$ will constitute an execution of Π, which we will call *view*. We are interested in showing that a property holds for an execution of a protocol Π. We represent this property by defining a set of functions $\mathsf{property}$ over $\textsc{exec}_{(\mathcal{Z},\mathcal{A})}[\Pi]$. If the property encoded in $\mathsf{property}$ holds in a particular *view*, we will have $\mathsf{property}(\mathit{view}) = 1$ and 0 otherwise. We are interested whether a property holds in all executions of a protocol, not only a particular one. We mean that for every $\mathsf{property}$ there exists a negligible function negl such that

$$Pr[\mathit{view} \leftarrow \textsc{exec}_{(\mathcal{Z},\mathcal{A})}[\Pi] : \mathsf{property}(\mathit{view}) = 0] < \mathsf{negl}(\lambda)$$

where negl is a negligible function in our security parameter λ.

A main advantage of using a system like UC is that it permits us to build protocols in a modular way. We will take advantage of this feature by building our protocols through components which we can swap depending on the needs. In particular, our construction will use consensus as a black box with certain properties. As long as those properties are accomplished by a consensus algorithm, we can insert it into our protocol. This structure gives the model flexibility as well as the ability to combine it with other consensus protocols.

3 The Proof-of-Work Chain

One of the fundamental aspects of the Bitcoin blockchain is the immutability that the proofs of work provide. While most of the attention over the proofs of work focuses on their role in consensus, the purpose they serve in securing the state of the chain is fundamental. The proofs of work coupled with the blockchain structure ensure that any change in a block can only be achieved by a considerable investment of computational power. If we want to change one block of the chain, the probability of constructing a new valid blockchain decreases exponentially in how many blocks follow after the changed block. Because a new chain will only be accepted by someone if it is at least as long as the one which that agent currently has, it becomes practically impossible to change the content inside of the chain. Security is compounded by the fact that the chain is constantly growing, making it even harder to catch up to. This property permits the users of Bitcoin to trust that their money will not suddenly disappear. The fact that blockchains can intrinsically generate trust has made them an interesting topic to study, as the ability to transfer the trust put in cryptography unto unknown agents has facilitated many things that were previously thought impossible.

Not everything about the proofs of work in Bitcoin is good. The process of finding a valid proof of work requires participants to brute force a hash function, a process colloquially known as mining, until a desired value is achieved. With Bitcoin's proofs of work (which we will refer to as Nakamoto proofs of work), if someone has more processors working on mining, they are more likely to find a valid hash. This fact is important because agents are incentivised to create blocks by receiving a fixed amount of Bitcoin for each block they generate. In practice, this incentive has led various investors to build dedicated mining facilities, undermining the distributed nature of the Bitcoin network. Because the difficulty of the mining process is (roughly) determined by the amount of participants, it now takes a considerable amount of computing power to maintain the network. Computation can only happen through electrical energy: it is estimated that the current power needed to maintain the Bitcoin network is close to the output of a medium-to-large nuclear reactor [Bit17]. Thus, there are already concerns over the sustainability of the Bitcoin network.

The assurance that no one can easily change the saved information is a desirable property for data storage, especially because it can engender trust between mutually untrusted parties. However, because of the possibility of parallelisation, these guarantees can only be maintained in a network with a considerable amount of computational power invested in it. The creation of trust can be achieved in one of two ways: in a sufficiently large network, like for Bitcoin, or by a deliberate investment of computational power by the parties. Due to these issues, the Nakamoto proofs of work do not fit in permissioned networks, which are considerably smaller than the Bitcoin network. Therefore, we would want to create an immutability guarantee that is independent of the size of the network. If we do not want the size of the network to affect the guarantee, then we need the computing power invested in a proof of work to not be subject to parallelisation.

The Nakamoto proofs of work are designed to function like a lottery, with each execution of the function acting as a ticket for the participant who called it. A lottery system encourages parallelisation, as having more cores computing proofs of work means having more tickets. As the proofs of work are the primary mode of consensus, the lottery system makes sense. However, if we separate the proofs of work from consensus process, we can avoid the lottery setting so the immutability guarantees are not linked to the incentive structure. We do this in order to avoid creating incentives for parallelisation. Even if the function cannot be parallelised, if we rely on any property of the output (besides it being properly computed) a participant could be motivated to compute multiple instances of the proof-of-work function. This is part of the reason why, in contrast with Nakamoto proofs of work, we will not use our proofs of work for consensus. Changing this fact means fundamentally altering the structure of the proofs of work, so we can choose to build them in such a way to realise the properties that we want.

When we speak of the immutability of the Bitcoin blockchain we speak of computing power, but the way it is reflected in practice is in time. The more computational power is invested in computing proofs of work, the less time it takes to find one. Therefore, we would like a way to encode the time spent during computation in a function and make it impossible to reduce this time by using several computational units to compute it. A way to prevent the use of parallel computations is by using a function that is inherently serial. It cannot be computed by separate processors as the result of the previous instance is necessary to start computing the next. This system is not enough, as it could be possible to analytically define the composition to avoid the iterated computation. To avoid this analytic shortcut, we add a permutation between every instance of the function. Not any function will work for this purpose, but we know at least one that will. Modular square roots composed with permutations provide a good candidate for these functions, as seen in Section 2.2. They can be adapted so that their computation takes a certain amount of clock time; they also provide a pseudorandom output, as seen in [LW15]. Thus, the proofs of work can additionally provide a public source of randomness for other purposes, something that has been explored for the Bitcoin blockchain with negative results [PW16].

The idea of serial proofs of work does not come without caveats, but the assumptions made are reasonable and in accordance to empirical evidence. The time spent for computing a function is a direct consequence of processor speeds. Therefore, the time spent in a computation cannot be strictly encoded without knowledge of the processor which computed it. We can avoid this issue if we consider the immutability guarantees over the strongest processor that could realistically be used for this purpose. Moore's law could suggest that the security of the blockchain might be undermined by the advance of processor technology. However, current technological design focuses on building multi-core processors instead of faster single cores. Because of the sequential nature of the proofs of work, they must be computed in one single core, making these advances irrelevant. We will later show that as long as the processors computing the proofs of work speed up at the same rate as the technological advances, earlier blocks with *weaker* proofs of work will still be immutable due to the blockchain structure.

3.1 Security of the Serial Proofs of Work

Most of the literature in blockchain protocols focuses on the security of the protocols themselves. The results regarding the security of the blockchain are a direct consequence of the protocol. The serial proof of work's primary purpose regards the structure of the blockchain itself and not the protocol. To prove the security properties provided by the blockchain structure, we will create a simple protocol with only one participant who is maintaining a personal blockchain. Due to the fact that the blockchain is maintained by exactly one person, there is no need for any consensus mechanism. Therefore, the security properties that we will prove in this chapter are intrinsic to the structure. This fact means that any existing protocol can incorporate serial proofs of work as a part of the protocol to acquire the security properties that we will prove.

The idea of serial proofs of work is innately related to time, so we need to define what we mean by time. In this work, we will consider time broken up in discrete time steps as if they were ticks from a clock. It is important to note that time steps explicitly represent the passage of time in the physical sense and not as something that can be affected by the computational power of the processors involved in the computation.

Our blockchain depends on two different functions that we will model as random oracles: hash functions and the serial proof of work. When a participant queries the random oracle with an input, it checks if it has been already queried with said input. If this is not the case, the oracle picks a number uniformly at random and outputs it to the participant, storing it in memory. If a participant queries a value that has already been queried, the oracle outputs the previously queried result instead. The first function we model in this way is the hash function $H(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^\lambda$, where λ is our **security parameter**. This model is the common way to represent hash functions in the literature due to the fact that hash functions should be *collision-resistant*, which means that it is very hard to find two distinct strings x and y such that $H(x) = H(y)$. Every participant can query this oracle a polynomial amount of times at every time step, and get the result immediately.

We are also interested in having a proof-of-work function that behaves as a random oracle. However, the modelling will not be as straightforward as for our hash function. Instead, we will define a process Golem to generate proofs of work which has access to a random oracle $\mathsf{PoW} : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$.[1] The random oracle PoW will have the same *lazy sampling* structure as H. This is a standard approach to proofs of work in the literature, although recently a new abstraction of the concept of a proof of work has been presented in [GKP17]. While we do not follow that construction, it is important to note that our proof of work is consistent with the abstraction presented in that work. Our proof of work will consist of a process that iterates PoW until it gets an instruction to stop. When the process stops, it will output the current output of the iteration, as well as a count of how many iterations were computed. However, we also want to represent the difference in computing power that different players have. This means that we will have not one process, but many of them, depending on each participant's **rate** γ. Intuitively, the rate represents the number of times that a participant can compute PoW sequentially in a time step. The computing power that each participant invests in computing the proof of work is encoded in γ. Note that the computing power in this case refers to the power of a single processor, as computing PoW sequentially is a process that cannot be parallelised. Therefore, we define a family of processes Golem_γ with $\gamma \in \mathbb{Q}^+$ which work in the following way:

[1]Note that both random oracle functions share the same parameter λ. This modelling choice simplifies the notation as well as the intuition. As a matter of fact, [LW15] suggests that it would actually be better to have λ considerably higher for PoW than for H (2048 versus 256). We can change the λ belonging to PoW by simply changing the hash function (or partitioning the information and concatenating the hashes of the partitions) used inside of Golem. This change will not affect anything else in the model but we will keep it as is to avoid adding another parameter.

```
$\mathsf{Golem}_\gamma$

On **input** start($x$): Set $s = 0$ and $y = H(x)$
    **Every** $^1/_\gamma$ **time steps**
        • $s \leftarrow s + 1$
        • $y \leftarrow \mathsf{PoW}(y)$
On **input** halt:
    • **Output** $(y, s)$
```

Note that this functionality does something that the participants are not allowed to do in other cases: it receives the result of a computation before the end of a time step. Whenever participants call PoW outside of $\mathsf{Golem}_{\gamma_j}$, they will get the result at the end of the time step, so they cannot emulate $\mathsf{Golem}_{\gamma_j}$ without calling it directly. We must set a lower bound on γ_j as otherwise a participant a_j could *speed up* the computation of $\mathsf{Golem}_{\gamma_j}$ by querying PoW directly instead of $\mathsf{Golem}_{\gamma_j}$ if $\gamma_j < 1$. Therefore, we will only work with $\gamma_j \geq 1$.[2]

Whenever $\mathsf{Golem}_{\gamma_j}$ is successfully executed for t time steps with input x, it will output $\left(\mathsf{PoW}^{\lfloor t/\gamma_j \rfloor}\left(H(x)\right), \lfloor t/\gamma_j \rfloor\right)$. In the rest of this thesis, we will abuse notation and write simply $\mathsf{PoW}(x)$ when we mean $\mathsf{PoW}\big(H(x)\big)$[3]. In cases where we are talking of running this protocol in a context where the rate is not relevant, we will refer to it simply as Golem. Note that s is the number of times that PoW was iterated, we will call s the **strength** of a proof of work. Each participant a_j in our model will have access to $\mathsf{Golem}_{\gamma_j}$ and will compute proofs of work by calling this process. In practice, if we would run the proof of work for ten minutes we would have iterated PoW for more than a hundred thousand times, so the supposition that at least one instance of PoW can be computed in a time step is a valid one.

The random-oracle model represents the fact that there is no way to shorten, predict or (effectively) precompute the computation. Note that any participant can query PoW a polynomial number of times in each time step, but that should not be enough to find shortcuts. A property that we want from this random oracle is pre-image resistance: if an agent has access to a value y in the range of PoW, he cannot find an element x in the domain such that $\mathsf{PoW}(x) = y$ in

[2]This modelling choice can be prevented by making it impossible for each participant to query PoW directly, however, this is not a natural constraint. On the other hand, it is possible to re-scale the size of the time steps in order to ensure that the rate will be fast enough, especially considering that the amount of iterations per minute is in the order of tens of thousands, according to [LW15].

[3]When we presented our proof-of-work function in Section 2.2, we mentioned that it included a permutation between every iteration of the modular square root. Our hash function H is *not* that permutation. In this model, the permutation is considered to be part of PoW. While it may be interesting to study the properties of this permutation, we will not do so in this work.

polynomial time with a non-negligible success probability (relative to the security parameter λ).

The proofs of work that the participants compute will be encoded in each block. However, to represent the proof of work computed over x we cannot simply write $\mathsf{Golem}_{\gamma_j}(x)$, as the process will output numerous distinct values depending both on how long the process was left running and the value of γ_j. Thus, to represent the proofs of work that have been computed, we will simply write $\big(\mathsf{PoW}^s(x), s\big)$.

An important aspect about our proofs of work is that while they take a long time to compute, it must be easy, and therefore quick, to verify them. Thus, every participant has access to a function $\mathsf{verifyPoW}$ which takes a triple (x, y, s) as an input and verifies whether $\mathsf{PoW}^s(x) = y$. Each participant can make polynomially many queries to $\mathsf{verifyPoW}$ and get the result at the end of the time step. This means that while computing multiple iterations of PoW takes time, the verification is considerably quicker. This *invertibility* is what caused us to choose the function presented in Section 2.2 as our candidate function. This function fulfills this characteristic, as reversing a square root is achievable by simply squaring the root. This is a fundamental part of the protocol, as we want the work to be time consuming to perform but easily verifiable.

First, we will define our setting and the components of our blockchain and then show that the immutability guarantees which we seek are indeed present. We will refer to these blockchains as proof-of-work chains or **PoW chains** as we will also deal with different blockchains in Chapters 4 and 5. As the name suggests, the PoW chain contains the proofs of work that the participant is continually computing. After a proof of work is completed, the participant builds a new block and computes the proof-of-work function over that block. We will name our only participant a_1, who will maintain the blockchain PC^1. We will refer to 1 as the index for a_1.

Our model for PoW chains requires a public-key infrastructure, as each block is signed by the participant who is computing it. This serves both as a way to prove ownership of the chain and as a security measure. We assume that the participant has access to an ideal signing functionality Σ which is unforgeable. The participant is assigned a public and secret key, pk_1 and sk_1 respectively, and may query the signing oracle to sign something ($\Sigma.\mathsf{sign}^1$) or to verify that a signature is valid ($\Sigma.\mathsf{verify}^1$). The participant may make polynomially many queries to either sign or verify a signature and get a response by the end of the time step. The signature scheme is not particularly relevant in this initial setting but will be in the following chapters.

Our PoW chains consist of signed blocks which contain pointers to the previous block in the chain, as seen in Figure 3.1. Formally, we define it as follows:

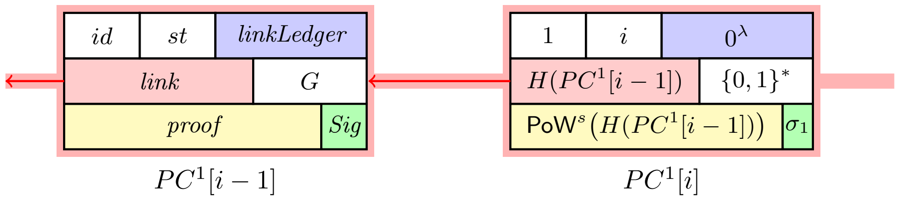

Figure 3.1: The proof-of-work chain maintained by a_1. The block on the left includes the names of the components while the one in the right represents how they are constructed. σ_1 represents the signature of the rest of the elements of the block by the a_1.

Definition 3.1 (PoW block)**.** *We say that $PC^j[i] = (st, id, link, linkLedger, G, proof, Sig)$ is the i-**th PoW block of** a_j where $B = (st, id, link, linkLedger, G, proof)$ if*

- *$st = i$ is the round when the block was created,*
- *$id = \{j\}$ is the index of the player who maintains the chain,*
- *$link \in \{0,1\}^\lambda \cup \{\perp\}$ is a hash,*
- *$linkLedger \in \{0,1\}^\lambda \cup \{\perp\}$ is the link to the previous ledger block,*
- *$G \subseteq \{0,1\}^* \cup \{\perp\}$ may contain additional information,*
- *$proof = (\mathsf{PoW}^s(link), s)$ for some $s \in \mathbb{Z}^+$*
- *$Sig = \{\Sigma.\mathsf{sign}^j(B)\}$ is a signature of the block by a_j.*

To refer to the first component of a block $PC^j[i]$, we will use the notation $PC^j[i].st$. We will use equivalent notation for every other component of the block.

Definition 3.2 (PoW chain)**.** *A **PoW chain of** a_j **for a genesis block** $PC^j[0]$, PC^j, is a sequence of PoW blocks $PC^j[0], \dots PC^j[p]$ where $B = (PC^j[0] = \Big(0, \{j\}, H(pk_j), \perp, \perp, \big(\mathsf{PoW}(H(pk_j)), 1\big)$*

$$PC^j[0] = \Big(0, \{j\}, H(pk_j), \perp, \perp, \big(\mathsf{PoW}(H(pk_j)), 1\big), \{\Sigma.\mathsf{sign}^j(B)\}\Big)$$

and there is a monotonous increasing sequence i_m with $i_0 = 0$ such that for all $PC^j[i_m]$ with $m > 0$ we have that $PC^j[i_m].link = H\big(PC^j[i_{m-1}]\big)$.

*Let $\mathsf{len}(PC^j)$ be the **length** of PC^j, that is, the amount of non-genesis blocks contained in the chain. We define $\mathsf{last}(PC^j)$ as the last block of PC^j (the one with the greatest st) and $PC^j[i, r)$ as the blockchain starting from $PC^j[i]$ until, but not including, $PC^j[r]$.*

Note that PoW blocks do not necessarily point to the block with the preceding st. This distinction is to allow the possibility of participants who temporarily stop running the protocol. Therefore, it is possible that $PC^j[i]$ does not exist while both $PC^j[i-1]$ and $PC^j[i+1]$ do. However, in this initial model this is not the case, so we will always have that $PC^j[i].link = H(PC^j[i-1])$ and therefore $\mathsf{len}(PC^1) = \mathsf{last}(PC^1).st$.

3.1.1 Proofs of Constant Strength

We wish to show that the serial proofs of work offer immutability in the sense that changing something done in the past requires an investment of time and therefore the ability to compute proofs of work faster. We will start in a setting where the strength of the proofs of work is always the same and a block is created in regular intervals called **rounds**. In this model, we will slightly simplify the structure of our blockchain. The component *linkLedger* of a block is not relevant to this particular model, so in this chapter we will assume that it will always be equal to 0^λ.

In this protocol $\mathsf{SingleLipwig}^\tau$, the participant a_1, who we will call the **maintainer** of the chain, will add a string G_i to the chain in every round i as $PC^1[i].G$. Each round will last $\tau + 1$ time steps in which $\mathsf{Golem}_{\gamma_1}$ will be running for τ of them and the last step is used to create the block. In this case, we will say that the round has **round time** τ. We will then prove that if a_1 wants to change G_i for an arbitrary string after the block has been created, she will not be able to do so without stopping the creation of new blocks.

Round i **of** $\mathsf{SingleLipwig}^\tau$

The round begins upon reception of the output from $\mathsf{Golem}_{\gamma_1}$

- Set $\sigma = \Sigma.\mathsf{sign}^j\big(H(PC^i[i-1]), 0^\lambda, G_i, \mathsf{PoW}^{\tau\gamma_1}(PC^1[i-1])\big)$
- Append $\big(H(PC^1[i-1]), 0^\lambda, G_i, \mathsf{PoW}^{\tau\gamma_1}(PC^1[i-1]), \sigma\big)$ to PC^1 as $PC^1[i]$
- **Query** for $\mathsf{Golem}_{\gamma_1}(PC^1[i])$
- After τ time steps, **input** `halt` to $\mathsf{Golem}_{\gamma_1}(PC^1[i])$

We do not present round 0 of the protocol, where a_1 gets a genesis block $PC^i[0]$ and calls $\mathsf{Golem}_{\gamma_1}$ with it as input for τ time steps. The clock is set such that $t = 0$ is the moment when a_1 queries for $\mathsf{Golem}_{\gamma_1}(PC^i[0])$. This means that the moment a_1 adds $PC^1[1]$ to her chain, $\tau + 1$ time steps have passed.

This model differs from the paradigm in blockchain modelling, as there is no adversary. This means that our definition of security cannot be based on an adversary trying to affect the execution of the protocol. The primary focus of this work is showing that the blockchain structure enhanced with serial proofs of work cannot be modified unless certain conditions are met. The immutability guarantees ensure that, as long as the protocol is running, no one can change the contents of the chain unless they have a faster processor. The process of changing the contents of at least one block and creating a new valid blockchain will be known as **rewriting the blockchain**. Whenever the new chain diverges from the original one will be known as the **rewriting point**. On the other hand, if at a certain point, the maintainer of a chain maintains more than one different copy of it, this will be known as a **fork**. The implication from the immutability guarantees is that not even the agent in charge of maintaining the blockchain can rewrite it unless certain conditions are fulfilled. On the other hand, a_1 can very easily fork the blockchain at any time.

If the maintainer of the blockchain is unable to change it, this permits an outside party to trust the contents of the chain without having to trust the person who created it. While the structure is not enough to show that the records are correct, an outsider can use the guarantees encoded in the proofs of work to know that something was registered at least a certain time ago. This is important because it prevents the maintainer of the chain from arbitrarily changing the contents of the chain. With this goal in mind, we will show that the serial proofs of work help to *time lock* the information encoded with it. We will first show that changing any block in the chain requires the re-computation of the proofs of work.

To be able to assert the following lemma we must also discuss the possibility of precomputation. There is nothing stopping the adversary from preemptively computing PoW with random inputs in the hope that one of those inputs will be valid. However, because the adversary can only make a polynomial amount of computations the probability of finding the right one is negligible. That being said, if the adversary starts with a random input and continuously computes PoW over it, she would gain a huge advantage in the case that she finds a useful input. There are ways to prevent this, like using a keyed permutation inside of PoW, which would slow down the verification but make it that any advantage achieved by precomputation is of only one instance of PoW. We do not explore this possibility in this model, as precomputation will only be useful with negligible probability, but it is something that could be useful in an implementation.

Lemma 3.3. *If H and PoW are random oracles with security parameter λ, suppose a_1 has a PoW chain PC^1 of length $i + k$ computed with rate γ_1 and constant round time τ. If a_1 changes the value of a block $PC^1[i]$ then he must change every block in $PC^1[i+1, i+k+1)$ as well and must compute $\mathsf{PoW}^{\lfloor \tau\gamma_1 \rfloor}$ k times.*

Proof. Suppose a_1 has changed $PC^1[i]$ to a (structurally correct) block PC^*. Note that it is not necessary to recompute the proof of work of $PC^1[i]$ if its contents are changed. Because $H(PC^1[i]) = H(PC^*)$ only with negligible probability, then we have that $PC^1[i+1].\mathit{link} \neq H(PC^*)$ except with negligible probability in the security parameter λ. Similarly, the advantage of $\mathcal{A}$ by precomputation is also negligible. Therefore she must change $PC^1[i+1].\mathit{link}$ and must change $PC^1[i+1].\mathit{proof}$ as well, by definition of PoW blocks. Changing $PC^1[i+1].\mathit{proof}$ implies computing $\mathsf{PoW}^{\tau\gamma_1}(PC^*)$. By induction on the blockchain, this fact is true for every block following $PC^1[i+1]$, of which there are k. Therefore, all blocks after $PC^1[i]$ must be changed and $\mathsf{PoW}^{\lfloor\tau\gamma_1\rfloor}$ must be computed k times. □

It is important to note that this proof is not for chains computed in $\mathsf{SingleLipwig}^\tau$ but is a general proof for all proof-of-work chains computed with constant rounds. This means that this property is a consequence of the PoW chain structure and not of any particular protocol. The fact that it is easy to verify proofs of work might suggest that it can be easier to first set $PC^1[i].\mathit{proof}[1]$ and apply the inverse of PoW^s, for some s, to it to choose *link*. While this means that we do not need to calculate PoW, the probability of finding a valid block PC^* such that $H(PC^*) = \mathsf{PoW}^{-s}(PC^1[i].\mathit{proof}[1])$ is negligible, as we require pre-image resistance from PoW. This means that any attempt to fork or rewrite the chain can only be achieved by doing the required work.

As opposed to Nakamoto proof of work, which provides a probabilistic argument to show the difficulty finding a valid block, serial proofs of work provide a deterministic guarantee of a lower bound on the time the block was created. The difference lies on the fact that finding a serial proof of work will always be achieved after the requisite time, which is not true of a Nakamoto proof of work. The security for Nakamoto proofs of work grows exponentially over the number of blocks but the relation to clock time is based on the number of processors computing the proof of work. The security in serial proofs of work grows only linearly but (unless there is a significant advance in processor technology) the time necessary to create blocks cannot be shortened. This means that an infinite amount of processors take no time to compute Nakamoto proofs of work but take as much time to compute serial proofs of work as the fastest processor. Therefore, security of serial proofs of work is directly related with real clock time, as building faster processors is considerably more complicated than building more of them. This fact implies that when we see a PoW chain, if we know the speed of the fastest processor available (be it available in general or to the particular agent maintaining the chain), we can be sure that the block has been created at a certain point in the past.

Definition 3.4. *Given a blockchain PC, we call a block $PC[i]$* ***time-locked by*** *S with respect to γ^* when the block had to be created at least S time steps ago.*

Theorem 3.5. *Let PC be a blockchain created in* $\mathsf{SingleLipwig}^\tau$ *with H and* PoW *parametrised by λ and rate γ_1 such that $\mathsf{len}(PC) = i + r$. Any block $PC[i]$ is time-locked by*

$$r(\tau\gamma_1/\gamma^* + 1)$$

if γ^* is the fastest rate that a_1 has access to, except with negligible probability in the security parameter λ.

Proof. By Lemma 3.3, to create a block with r blocks *in front of it*, it is necessary to compute the proofs of work in the following r blocks. Because of the serial nature of the proofs, this can only be achieved by calling Golem for r times in a row. The only way to avoid doing this computation is by finding a collision in the random oracles, which has negligible probability. Because a_1 has access to $\mathsf{Golem}_{\gamma^*}$ then every call would take ${}^{\tau\gamma_1}/_{\gamma^*}$ time steps. Because a_1 must make r calls to it in sequence and making a valid block takes one time step while $\mathsf{Golem}_{\gamma^*}$ is not running, the block must have been created at least ${}^{\tau r\gamma_1}/_{\gamma^*} + r$ time steps ago. $\square$

We have shown that the structure of the blockchain itself can serve to timestamp the information encoded inside of it. This is true of the blockchain regardless of the way it was created. In practice, this property is important because it ensures that changes to the blockchain can only happen with a faster processor and after a certain time has passed. To model this situation, we will create the following game between a_1, who acts as a prover, and a verifier $\mathcal{V}$.

Definition 3.6. *We define the **round-based Prover-Verifier (PV) game** for a blockchain protocol Π between a participant $\mathcal{P}$ with rate γ and a verifier $\mathcal{V}$ as follows:*

- *The verifier outputs a triple of integers (i, r, s), with $i > 0$ and $r, s \geq 0$ to the prover*
- *At the end of round $i + r - 1$, with $r \geq 0$, $\mathcal{V}$ outputs a uniformly random string $x \in \{0, 1\}^\lambda$ to $\mathcal{P}$*
- *At the beginning[4] of round $i + r + s$ the prover sends a blockchain BC to $\mathcal{V}$. The prover wins if the blockchain is correctly constructed according to Π, x is encoded in block $BC[i]$ and $\mathsf{len}(BC) = i + r + s$*

This game represents the situation where the party maintaining the chain must covertly change something in a block that is r blocks deep. The prover will win the game if she[5] successfully creates a blockchain where x is encoded in a block that is $r + s$ blocks deep. In this game, the prover can only make a polynomial amount of computations, and therefore a polynomial amount of forks. Because the prover cannot predict the value x (and cannot create 2^λ forks in which each block has a different value of x), this means that the prover must rewrite the blockchain in order to do it. However, the prover has only a limited time to do so. More importantly, this cushion of time that the prover has to rewrite the blockchain must also be represented in the rewritten blockchain. The verifier knows the rate of the protocol, but the prover may have access to

[4]After one time step, so she has the opportunity to create the block

[5]For this game, we will use female pronouns for the prover a_1 and male pronouns for the verifier $\mathcal{V}$.

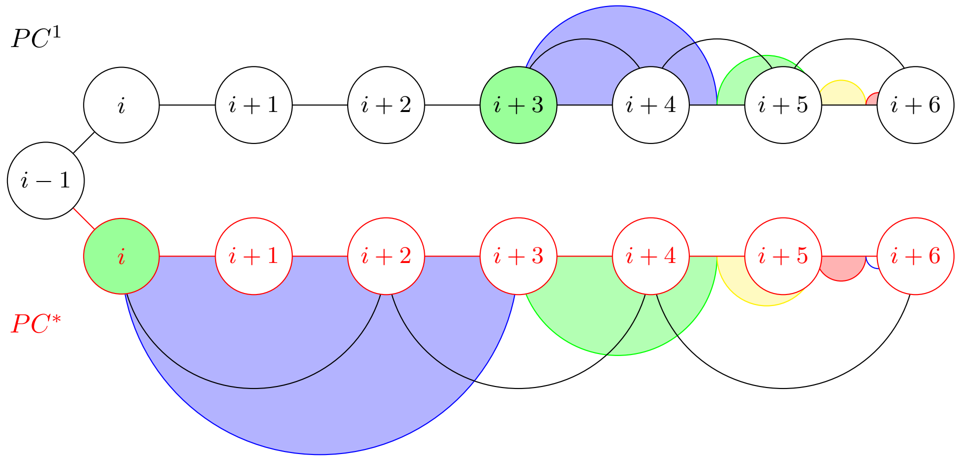

Figure 3.2: An example of the round-based PV game with $r = 3$ The prover has rate $\gamma^* = 2\gamma_1$. The bottom chain is the rewritten one while the one on top represents the expected chain. Each colored semicircle represents an interval of time; the semicircles with the same colors happen at the same time. When the rewritten chain catches up to $i + 3$, the original chain has added a new block that the rewritten chain has to catch up to. The rewritten chain is long enough by round $i + 6$, meaning that $\mathcal{P}$ can only win the game if $s \geq 3$

a stronger rate. The ability of the prover to win a game is determined by r but also by the rate she might have access to, γ^*. For SingleLipwig we know that by Theorem 3.5 the prover will not be able to arbitrarily rewrite the chain. However, we can strengthen the result by finding values for r and s. It is easy to see that unless $r = 0$, if the prover can only compute proofs of work at the rate of the protocol, she will never be able to win.

Lemma 3.7. *Suppose $\mathcal{P}$ and $\mathcal{V}$ are playing the round-based PV game for* $\mathsf{SingleLipwig}^\tau$ *with rate γ. If H and* PoW *are random oracles with security parameter λ, if $\mathcal{P}$ only has access to* Golem_γ *then she can only win if $r = 0$*

Proof. For $\mathcal{P}$ to win the game, she needs to create $r + s$ blocks in s rounds. By Theorem 3.5 it takes $r + s$ rounds for $\mathcal{P}$ to create these blocks. Therefore, she can only win the game if $r + s = s$. □

Lemma 3.8. *Suppose a_1 is running* $\mathsf{SingleLipwig}^\tau$ *with verifier $\mathcal{V}$ and H and* PoW *are random oracles with security parameter λ. Suppose that at time step $i + r$, a_1 becomes able to query* PoW_{γ^*} *with $\gamma^* > \gamma_1$ and starts doing so in* $\mathsf{SingleLipwig}^\tau$. *If at the same time a_1 starts rewrites the block $PC^1[i]$ at round $i + r$, the new chain PC^* will always have $\lfloor r\gamma_1/\gamma^* \rfloor$ blocks less than the original PC^1*

Proof. Now that a_1 has access to a faster processor, every new block will have stronger proofs of work, as they will contain proofs with $\tau\gamma^*$ iterations of PoW instead of $\tau\gamma_1$. However, the old blocks will continue to have strength $\tau\gamma_1$. In particular, blocks in $PC^i[i+r,)$ will have strength $\tau\gamma_1$. This means that the new chain will catch up to block $PC^1[i+r]$ in $r\tau\gamma_1/\gamma^*$ time steps. During this time, the main chain will have added $\left\lfloor \frac{r\tau\gamma_1/\gamma^*}{\tau} \right\rfloor = \lfloor r\gamma_1/\gamma^* \rfloor$ blocks. Starting from this point, we have the same situation than in Lemma 3.7, meaning that the distance between the main chain and the new chain will always be of $\lfloor r\gamma_1/\gamma^* \rfloor$ blocks. □

The serial proofs of work give us a guarantee that if the participant did not start computing a fork at the same time as she adds the blocks that she wants rewritten, she is not able to create a rewritten chain that is as long as the original one. This is only true, however, as long as she is not able to calculate proofs of work in a quicker way. Access to a faster processor will mean that she will eventually catch up to the chain; however, it implies an investment of time. We will define a notion of security to characterise this property.

Definition 3.9. *Given random oracles H and* PoW *and the execution of a round-based protocol Π with verifier $\mathcal{V}$, we say a blockchain BC is $\theta(\gamma^*, r)$-**secure** if a prover with access to* $\mathsf{Golem}_{\gamma^*}$ *cannot win the round-based PV game for $s = \theta(\gamma^*, r)$. We say BC is **totally secure** if the prover cannot win the game for any s given $r > 1$.*

Theorem 3.10. *Let H and* PoW *be random oracles with security parameter λ. A chain created by* $\mathsf{SingleLipwig}^{\tau}$ *with rate γ is $\theta(\gamma^*, r)$-secure with*

$$\theta(\gamma^*, r) = \left\lceil \frac{r\gamma}{\gamma^* - \gamma} \right\rceil - 1$$

Proof. We are interested in showing that a prover cannot win the PV game when $s \le \frac{r\gamma_1}{\gamma^* - \gamma_1}$. We have already proven in Lemma 3.7 that if $\gamma^* \le \gamma$, then the prover can never win, so then the chain is totally secure.

In order for $\mathcal{P}$ to win the game, she needs to be able to compute $r+s$ proofs of work in the time between getting x (round $i+r$) and having to output the blockchain containing it (round $i+r+s$). Therefore, she has s rounds to compute $r+s$ proofs of work. Because she has access to $\mathsf{Golem}_{\gamma^*}$, each proof of work takes her γ/γ^* of a round. Therefore, she needs that

$$s \ge (r+s)\frac{\gamma}{\gamma^*}$$

Therefore, $\mathcal{P}$ can only win the game when the following holds:

$$s \ge \frac{r\gamma}{\gamma^* - \gamma} - 1$$

This means that if s is less than this quantity she cannot win, so the chain is secure.

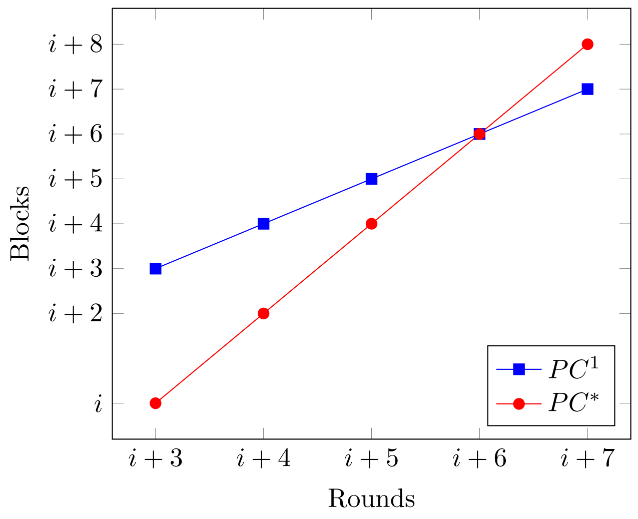

Figure 3.3: This plot represents an attempt rewriting the i-th block, starting in round $i+4$. The proofs of work in the rewritten chain PC^* are being computed with a rate of $\gamma^* = 2\gamma_1$. In round $i+8$, the rewritten chain becomes long enough to be acceptable.

We can also prove this without explicitly using the PV game: We know that every round is comprised by τ time steps. We know that PoW_{γ^*} computes proofs of work in $\tau\gamma/\gamma^*$ time steps and $\gamma/\gamma^* \leq 1$. This means that for every block that should be added to the chain, the prover will have $\tau - \tau\gamma/\gamma^* = \tau(\gamma^*-\gamma/\gamma^*)$ time steps each round to compute the proofs of work for the r blocks that she needs to catch up. To compute the necessary proofs of work $\mathcal{P}$ needs $r\tau\gamma/\gamma^*$ time steps. Due to the advantage every round, she takes

$$\frac{r\tau\gamma_1/\gamma^*}{\tau(\gamma^*-\gamma_1/\gamma^*)} = \frac{r\gamma_1}{\gamma^* - \gamma_1}$$

rounds to do so.

For a visual example, Figure 3.2 represents this process for $\gamma^* = 2\gamma_1$ and $r = 3$. The graph in Figure 3.3 shows the difference of the rates and how it permits the rewritten chain to catch up. □

We have now shown not only that a faster rate is necessary to rewrite the blockchain, but we have also found how much time would someone need to be able to do so without disrupting the protocol.

3.1.2 Proofs of Variable Strength

Previously, we assumed that proofs of work were always of the same strength, determined by the length of the round, which was fixed in the beginning. We will relax this assumption by giving a_1 the ability to stop the computation of PoW at (almost) any time. This frees a_1 from the rigid round structure and permits him to create blocks as she sees fit. There will be some limitations though: while in simpleLipwig we had a parameter τ determining the length of a round, this new protocol will have a parameter ω which will fulfill a similar role, acting as a lower bound. However, ω will not set a bound for the minimum number of time steps, as τ does, but for the strength of the proofs of work that will be added to the blocks. Indirectly, this will in turn create a lower bound on the time spent to create each block. Similarly to SingleLipwig, the protocol starts by calling $\mathsf{Golem}_{\gamma_1}$ on the genesis block, until its output will have strength ω.

Round i of $\mathsf{SingleVarLipwig}^{\omega}$

The round begins when a_1 gets an output W_{i-1} from $\mathsf{Golem}_{\gamma_1}$

- **Set** $t = 0$
- Set $\sigma = \Sigma.\mathsf{sign}^j\big(H(PC^i[i-1]), 0^\lambda, G_i, W_{i-1}\big)$
- Append $\big(H(PC^1[i-1]), 0^\lambda, G_i, W_{i-1}, \sigma\big)$ to PC^1 as $PC^1[i]$
- **Query** for $\mathsf{Golem}_{\gamma_1}(PC^1[i])$

Each time step

- Increase t by 1

If $t\gamma_1 \geq \omega$

- a_1 may input `halt` to $\mathsf{Golem}_{\gamma_1}$

The main difference in this protocol is the ability of a_1 to choose when to stop the round. This, in turn, implies that the strength of the proofs of work encoded in the blocks will differ from one block to the next. This introduces some changes in the proofs, but mostly the structure remains the same. In this setting, we are interested in the strength of a given proof of work. Therefore we will define a function str, which given a proof of work will output its strength: $\mathsf{str}(\mathsf{PoW}^s(x), s) = s$. In certain cases we will speak of the strength of a PoW block, in which case we mean the strength of the proof of work included in the block. We will abuse notation and apply str to a block $PC^1[i]$, to refer to the strength of $PC^1[i].\mathit{proof}$.

It might not be entirely clear why SingleVarLipwig requires a parameter ω to act as a lower bound for the strength of the proofs of work. Is there a difference between a block of strength 10 and ten blocks of strength 1? The answer lies in the time that $\mathsf{Golem}_{\gamma_1}$ is not running. In each round of the protocol, there is a time step in which a_1 is creating a new block. That time step is not represented in the proof of work, as no work is being done there. In general, this time step of *wasted* time is not important, as rounds are expected to be considerably longer than one time step. If we compare to other blockchains like Bitcoin, with rounds of approximately ten minutes, each round will have more than tens of thousands of time steps. Therefore, that one time step in which $\mathsf{Golem}_{\gamma_1}$ is not active is not important. However, if we do not demand a minimum proof of work strength, a_1 could create rounds that last two time steps, meaning that for every time step spent computing there is a time step where no computation is done. This implies that only half of the total time is spent computing proofs of work. For the immutability guarantees to work as we expect them to, we want to minimise the amount of time in the protocol where a proof of work is not being computed. This fact will affect our modelling decisions in the following two chapters.

First, we will prove that we can also timestamp blocks in this setting. As we do not have rounds anymore, we have to focus on the strength of the protocol.

Theorem 3.11. *Let PC^1 be a blockchain created in* SingleVarLipwig *with H and* PoW *parametrised by λ such that $\mathsf{len}(PC^1) = i + r$. Any block $PC^1[i]$ is time-locked by*

$$r + \sum_{k=1}^{r} \left\lceil \frac{\mathsf{str}\big(PC^1[i+k]\big)}{\gamma^*} \right\rceil$$

if γ^* is the fastest rate that a_1 has access to, except with negligible probability in the security parameter λ.

Proof. We will prove this by induction on the distance from the last block. Clearly $\mathsf{last}(PC^1) = PC^1[i + r]$ could have been created at any time, as it has no blocks in front of it. Therefore, our base case holds.

Suppose that the block $PC^1[i + 1]$ is time-locked by

$$r - 1 + \sum_{k=1}^{r-1} \left\lceil \frac{\mathsf{str}\big(PC^1[i+1+k]\big)}{\gamma^*} \right\rceil$$

Because $PC^1[i+1]$ has a pointer to $PC^1[i]$ then it had to be created at least one time step after it. More importantly, because it has a proof of work of strength $\mathsf{str}(PC^1[i + 1])$ computed over a pointer of $PC^1[i]$, the shortest time needed to compute it is $\left\lceil \frac{\mathsf{str}(PC^1[i+1])}{\gamma^*} \right\rceil$ plus the one time step needed to build the block. Therefore, $PC^1[i]$ must have been created at least

$$r - 1 + \sum_{k=1}^{r-1} \left\lceil \frac{\mathsf{str}\big(PC^1[i+1+k]\big)}{\gamma^*} \right\rceil + 1 + \left\lceil \frac{\mathsf{str}\big(PC^1[i+1]\big)}{\gamma^*} \right\rceil$$

time steps ago, which is lesser or equal than

$$r + \sum_{k=1}^{r} \left\lceil \frac{\mathsf{str}\big(PC^1[i+k]\big)}{\gamma^*} \right\rceil$$

□

In order to prove similar security guarantees as the ones in the previous model, we will need a similar game. However, as we lose the round structure, we have to be more flexible and worry about time steps instead of rounds.

Definition 3.12. *We define the **Prover-Verifier (PV) game** for a blockchain protocol* Π *between a participant* $\mathcal{P}$ *with rate* γ *and a verifier* $\mathcal{V}$ *as follows:*

- *The verifier outputs a triple of integers* (t_0, t_1, t_2) *with* $0 < t_0 < t_1 < t_2$ *to the prover*
- *At time* t_1*,* $\mathcal{V}$ *outputs a uniformly random string* $x \in \{0,1\}^\lambda$ *to* $\mathcal{P}$
- *At time* $t_2 + 1$*, the prover sends a blockchain* BC *to* $\mathcal{V}$*. The prover wins if* x *is encoded in a block* $BC[i]$ *and* $\sum_{k=i+1}^{\mathsf{len}(BC)} \left(\mathsf{str}(BC[k])/\gamma + 1\right) \geq t_2 - t_0$

Note that the index of the block $BC[i]$ is not important, the only thing that is relevant is whether the block is time-locked by $t_2 - t_0$ according to γ. Similarly, we need a new definition for a chain to be θ-secure.

Definition 3.13. *Given random oracles* H *and* PoW *and the execution of a protocol* Π *with verifier* $\mathcal{V}$*, we say a blockchain* BC *is* $\theta(\gamma^*, t)$***-secure** *if a prover with access to* $\mathsf{Golem}_{\gamma^*}$ *cannot win the PV game for* $t_2 = \theta(\gamma^*, t_1)$*. We say* BC *is **totally secure** if the prover cannot win the game for any* t_2 *given* $t_1 > 1$*.*

Theorem 3.14. *Let* H *and* PoW *be random oracles with security parameter* λ*. A chain* PC^1 *created by* $\mathsf{SingleVarLipwig}^\omega$ *with rate* γ *is* $\theta(\gamma^*, t_1)$*-secure where*

$$\theta(\gamma^*, t_1) = \frac{\gamma^* t_1 - \gamma(t_0 + 1)}{\gamma^* - \gamma} - 1$$

Proof. To win the game, the prover $\mathcal{P}$ needs to be able to compute proofs of work such that $\sum_{k=i+1}^{\mathsf{len}(BC)} \left(\mathsf{str}(BC[k])/\gamma + 1\right) \geq t_2 - t_0$. Because changing the number of blocks does not affect this, she needs simply to have this

$$\mathsf{str}(PC^1[i+1])/\gamma + 1 \geq t_2 - t_0$$

where x is encoded in block i.

In other words, she needs to be able to compute a proof of work of strength $(t_2 - t_0 - 1)\gamma$ in the time between t_1 and t_2. Therefore, she needs the following

$$(t_2 - t_0 - 1)\frac{\gamma}{\gamma^*} \leq t_2 - t_1$$

Following what we did in Theorem 3.10, this implies that for the prover to win, she needs

$$t_2 \geq \frac{\gamma^* t_1 - \gamma(t_0 + 1)}{\gamma^* - \gamma}$$

□

While having variable proofs of work introduces more complications to the functioning of the blockchains, they present a more realistic setting while maintaining most of the properties we desire from the PoW chains.

4 An Idealised Model

In the previous chapter we presented the PoW chain in a setting with a single agent. This is not the standard setting for a blockchain, usually blockchains act as ledgers maintained by all participants in a network. In this chapter and the next we will present a blockchain in this context, where it acts as a distributed ledger of a cryptocurrency. However, the idea of each participant owning a personal PoW chain will be kept, with the ledger chain inheriting the immutability guarantees from each individual PoW chain. While most of the literature on blockchains centers on the consensus protocol, our work focuses on the structural properties of the blockchain and therefore uses consensus as a black box. We will work on a permissioned setting, which makes reaching consensus considerably simpler and of less interest. We do this to highlight the structural properties of the serial proofs of work, but this does not mean that serial proofs of work are incompatible with a permissionless setting.

To model our protocol $\mathsf{IdealLipwig}^{\mathcal{T}}$, we present the following properties of our setting:

Interactive Turing Machines We will work in a standard Interactive Turing Machine (ITM) model where every participant is spawned and controlled by the environment $\mathcal{Z}$ and an adversary $\mathcal{A}$, which are probabilistic polynomial-time algorithms. There is a global clock that all participants have access to.

Time There are two distinct aspects of time in our protocol: time steps and rounds. Time will be divided in *time steps*, in which each participant can make a polynomial amount of computations. Each time step corresponds to a tick of the global clock. Additionally, every message sent will arrive by the end of the time step, meaning that we are working in a synchronous setting. On the other hand, our protocol will be based in rounds, indexed by $i \in \{1, 2, \dots\}$, each of which will correspond to a block being added to all the blockchains. In $\mathsf{IdealLipwig}^{\mathcal{T}}$, each round will last a fixed amount of time steps, bookended by the execution of the Golem protocol. A round will start with a call to Golem and end whenever Golem outputs a result. In contrast with other protocols, the length of rounds is set at the beginning of the protocol and is not a consequence of block creation. Later, we will relax this assumption on the length of the round and will allow them to vary in length.

Adversary We will represent the percentage of honest players by $Q \in [1/2, 1]$. At the beginning of the protocol, at most $1 - Q$ percentage of the total participants will be considered to be corrupt. These parties are controlled by an adversary $\mathcal{A}$ who determines their every action. These participants do not need to follow the protocol and can interact with $\mathcal{Z}$ in ways outside of what is prescribed by the protocol. At any time during the protocol, the adversary can corrupt a participant (as long as there are less than $(1 - Q)n - 1$ corrupt participants) by issuing a message $\texttt{corrupt}(j)$ to $\mathcal{Z}$. However, this corruption is not immediate and the adversary will only gain control of the new node in the next round. This means that while the corruption of participants is dynamic, it can be seen as static *within* each round. Once a participant is corrupted, it cannot become honest again.[1]

Due to the structure of our protocol, attempts to fork the blockchain can be identified by any honest participant. Therefore, we wish to make a distinction between participants who are under the control of the adversary (which we will call adversarial) and the participants that are actively attempting to undermine the protocol in such a way that they are discovered, which we will refer to as **antagonistic**. The adversary is interested in not appearing as antagonistic, as that will imply losing power in the form of honest parties removing antagonistic parties from the protocol. This implies that the protocol is protected not only by the security guarantees but also because the adversary is incentivised to cooperate with honest players and take part in the protocol in an honest way.

Network Assumptions In this first model, we assume the network contains a fixed number of participants, n, and that no participants can be added to the network after the start of the protocol. These participants are always active and participating in the protocol. In Chapter 5, we will weaken both of these assumptions, allowing participants to join during the execution of the protocol and letting them stop participating temporarily. All participants are connected to each other, so the network is perfectly connected and every message arrives immediately to all the recipients. To simulate this in our abstraction, every participant outputs a message to the adversary $\mathcal{A}$ who must then deliver the messages to every participant. The adversary must deliver the messages within the same time step, but can deliver them in any order she pleases. All communication by honest parties is directed to all participants, but the adversary is able to send

[1]For clarity in the presentation, we will refer to the adversary using female pronouns, while we use male pronouns for the participants.

messages to a subset of the participants without them (directly) realizing that it was not broadcast to everyone.

The network in which we will present our model is a **permissioned** one. This means that the number and identity of the participants is known by all of them. This allows for simpler and more efficient consensus mechanisms, as well as more favorable network properties. This does not entail that the serial proof of work only has a place in such a network. The reality is actually the opposite, as the investment of computational power and time to maintain PoW chains help prevent Sybil attacks, the bane of permissionless networks. This seems to suggest that there is a possibility to bootstrap a permissionless model from the permissioned one, in the style of [PS16b], but this goes beyond the scope of this work. Our choice on permissioned networks also allows the existence of a particular feature in our model: the expulsion of misbehaving agents.

Public-Key Infrastructure At the beginning of the protocol, the environment $\mathcal{Z}$ spawns n participants a_j and assigns a public and a secret key to each one, pk_j and sk_j respectively. Every public key is known to all the participants as well as the relation between keys and identities. Each participant a_j has access to the ideal signing functionality Σ through the queries $\Sigma.\mathsf{sign}^j$ and $\Sigma.\mathsf{verify}^k$ for $k \in \{1, \ldots, n\}$. The signature scheme is assumed to be unforgeable, so $\mathcal{A}$ cannot forge any signature or find the secret keys belonging to uncorrupted participants. All communication between participants will be authenticated by signatures, so participants will always know who sent a certain message. Whenever the adversary corrupts a party a_j, it gains access to the functionality $\Sigma.\mathsf{sign}^j$ and may sign any message with that key, even if it does not originate from player a_j.

Random Oracles This model will use the same random oracles H and PoW presented in Section 3.1. Participants will still interact with PoW through Golem, as in the previous chapter.

Transaction Pool To simplify the choice of transactions, every participant is running an instance of a protocol $\mathsf{TxsPool}$. In every time step, the environment $\mathcal{Z}$ inputs a (possibly empty) set of valid transactions to the instance of $\mathsf{TxsPool}$ of all participants. Participants can query $\mathsf{TxsPool}$ in two ways: they can input `receive`, in which case they get a set of all transactions contained in $\mathsf{TxsPool}$ or they can input `clean` as well as a list of transactions. In this case, the set of transactions are eliminated from $\mathsf{TxsPool}$. The transactions will be generated by the environment, simulating the creation of transactions by users of a cryptocurrency. In practice, there is a possibility that two transactions conflict with each other, as someone might try to spend the same money twice. In this case, at most one of the transactions could be added to the ledger. For simplicity, no conflicting transactions will be generated, which means that every transaction should be added to the blockchain. This assumption works to simplify the security proof of the protocol and can be removed without consequences for the modelling. To simplify notation, we will refer to $\mathfrak{T}$ as the **transaction space**, that is, every valid transaction is contained in the set.

Participant Indices We will work in a network with n participants a_j, who will be identified by the index j. In similar models, the identity of the participants will correspond to their public key, with any participant being able to have multiple identities by having multiple public keys. However, a permissioned setting guarantees that there is a one-to-one correspondence between keys and participants. More importantly, it allows us to have a link between indices and keys that can exist in practice and not just in the abstraction. Each participant will have an index associated to them and every player will know which index corresponds to which identity (and public key). This is especially relevant because the indices induce an ordering over the players. We will take advantage of this ordering to avoid relying on consensus protocols to order the signatures and proofs of work that will be added to the blocks. Whenever we speak of proofs of work or signatures being *ordered* we refer to the order induced by the indices: that is, the signature of participant a_j comes before the signature of participant a_{j+1}. For modelling purposes, we will also have a function ind which takes as input a set of proofs of work or signatures and outputs the indices of the participants who issued them. The set of the indices of all players will be represented by $N = \{1, \ldots, n\}$. Our protocol includes the possibility of misbehaving players being removed from the protocol, so the set of participants might change during the execution. The current set of active indices will be encoded in each block, which also allows participants to access this list during the execution of the protocol.

4.1 Components and Definitions

This model combines the standard model of a blockchain as a distributed ledger that is maintained by multiple agents with the models presented in Chapter 3, where a PoW chain is maintained individually by each participant. In the following model each participant will keep two chains: the ledger chain, which is maintained by the network, and their own personal PoW chain. While the ledger chain works in the same way as a traditional blockchain, with participants agreeing on the next block to be added, each participant has their own PoW chain. After a proof of work is completed, the participant builds a new block and computes the proof-of-work function over that block. These chains run parallel to the main ledger chain and are linked to it by hash pointers between each block. While PoW chains are maintained by only one participant, they do not have to be kept private. This means that every change in any PoW block can be recognised by looking at the ledger chain and vice versa.

There is a natural question that follows this construction: *Why have the PoW chains in the first place? Why not do the proof of work over the main chain instead?* The primary concern is that our security is based on time needed to compute the proofs of work. Therefore, any wait time between executions of the proof-of-work function will weaken the overall strength of the proof. Because PoW chains do not depend on what the network does, participants can (almost) immediately start computing the next instance of the function after finishing the previous one. Therefore, we do not have to worry about any delays in the creation or reception of the block, as we would do if we were working over the ledger chain. In the previous chapter we saw that in every round, there is one time step where the proof-of-work function is not running. This is not a problem, as that one time step is always necessary when creating blocks. However, if there was more time where the proof-of-work function is not being computed, this wait time could be exploited to rewrite the chain. Using personal PoW chains instead of waiting for a block built by consensus ensures that no time will be lost this way. As an additional advantage, PoW chains provide another dimension to the blockchain by providing each agent with a personal chain that inherits the security guarantees of the ledger chain and therefore the whole network. In this context, all PoW chains have the same level of security, but this will not be the case later. Additionally, the use of PoW chains permit us to build an ecosystem of blockchains which compound their security in a stronger network, as explored in Chapter 6.

4.1.1 The Ledger Blockchain

Having defined the PoW chains in the previous chapter, we will focus on the definition of the blockchain which fulfills the traditional role, the ledger chain. In general, the ledger chain is a "bigger" PoW block, containing multiple proofs of work and signatures while PoW blocks contain only one. An important difference is that in this chain the link to the PoW chains is found with the proofs of work, not as an independent component. While they have a similar structure, the ledger chain is more rigid, as a stricter structure is desirable for something that is maintained by the entire network instead of by each agent individually. To simplify the definition, we must first define the concept of a signature space.

Definition 4.1 (Signature space of M)**.** *Suppose we have n participants in the protocol, denoted by a_j with $j \in N$ and let there be a set of bitstrings $M \subseteq \{0,1\}^*$, we call $\Sigma\big(M\big)$ the* ***signature space of*** *M if*

$$\Sigma\big(M\big) = \{\Sigma.\mathsf{sign}^j(m) \mid j \in N, m \in M\}$$

Similarly, we define $\Sigma_J\big(M\big)$ with $J \subseteq N$ as

$$\Sigma_J\big(M\big) = \{\Sigma.\mathsf{sign}^j(m) \mid j \in J, m \in M\}$$

In the case $|J| = \{j\}$ we simply write $\Sigma_j\big(M\big)$.

Given that definition, we can continue to define our ledger chain.

Definition 4.2 (Ledger block)**.** *We say that $BC[i] = (st, NA, link, T, G, P, Sigs)$ is a **ledger block** when*

- *$st = i$ is the round where the block was created,*
- *$NA \subseteq \mathbb{N}$ is the set of indices of the active participants,*
- *$link \in \{0,1\}^\lambda \cup \{\bot\}$ is the link to the previous block,*
- *$T \subseteq \mathfrak{T}$ is an ordered list of transactions,*
- *$G \subseteq \{0,1\}^*$ encodes proofs of cheating*
- *$P \subset \{0,1\}^\lambda \times \{0,1\}^\lambda \times \Sigma\big(\{0,1\}^\lambda \times \{0,1\}^\lambda\big)$ is an ordered set of triples, such that if $(x, y, z) \in P$ then there exists $j \in NA$ such that $z = \Sigma.\mathsf{sign}^j(x, y)$,*
- *$Sigs \subseteq \Sigma\{(st, NA, link, T, G, P)\}$ is an ordered set of signatures of the tuple $(st, NA, link, T, G, P)$.*

In the same way as *proof* in the PoW chain, P contains the proofs of work. However, here the proofs of work are represented as triples consisting of the proofs themselves, hash pointers of the blocks in which they are contained and a signature by the maintainer of that particular PoW chain. In contrast, G fulfills a very different purpose in the ledger chain. Here we will record the attempts of the adversary to fork the chain. Every time there are two blocks from the same round signed by the same participant, a pointer to those blocks will be added to G in the ledger block. This will permit participants to record malfeasance in the chain itself, creating an immutable record of this.

Definition 4.3 (Ledger chain)**.** *A **ledger chain with genesis block** $BC[0]$, BC, is a sequence of ledger blocks $BC[0] \dots, BC[p]$ where for every $i > 0$ we have that $BC[i].link = H(BC[i-1])$ and*

$$BC[0] = \Big(0, N, \bot, \bot, \bot, Ps, \Sigma\big(0, N, \bot, \bot, \bot, Ps\big)\Big)$$

where N is the set of indices of all participants initialised before the start of the protocol and $Ps = \{(\bot, H(PC^j[0]), \Sigma.\mathsf{sign}^j(\bot, H(PC^j[0]))) \mid j \in N\}$.

*Let $\mathsf{len}(BC) = p$ be the **length** of BC. We define $\mathsf{last}(BC) = BC[p]$ as the last block of BC and $BC[i, r)$ as the blockchain starting from $BC[i]$ until, but not including, $BC[r]$. To define the ledger encoded in the chain, we will write $BC.T$*

We have connected the ledger chain with the PoW chains through the proofs of work in P. However, we want the connection to go both ways, for that we will use the component *linkLedger* which we had ignored in the previous chapter. In every PoW block, *linkLedger* will contain a hash of the ledger block that was

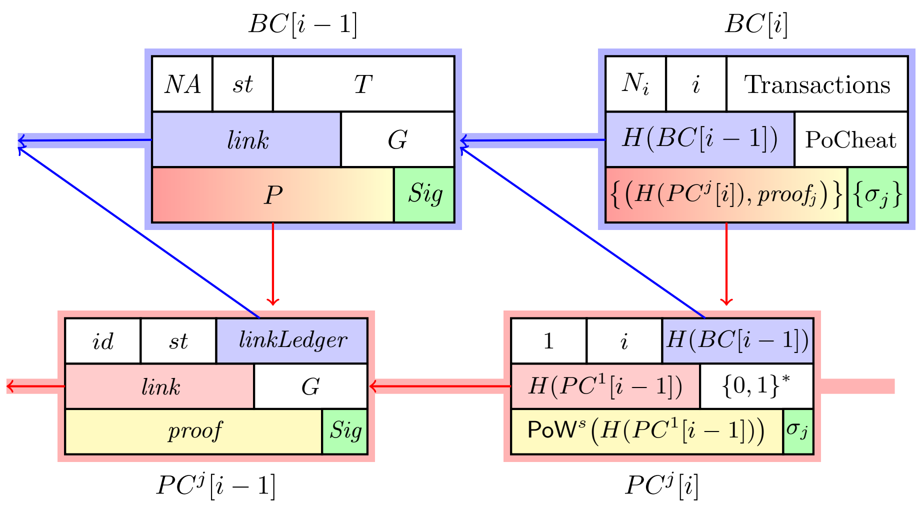

Figure 4.1: The blue ledger chain and the red PoW chain. Blue components point to the ledger chain and red ones point to the PoW chain. The proofs of work can be found in the *proof* component of the PoW blocks and in P for the ledger blocks.

computed the previous round. This means that the content of each block in the ledger chain will be reflected in every proof of work, albeit in an indirect sense. This will entail that a change in the ledger chain will forcibly imply a change in the PoW chain as well, meaning that the work for PoW must be done again. There are many properties that must be fulfilled for a PoW chain and a ledger chain to be connected, as we will see now.

Definition 4.4. *A PoW chain PC^j, is* ***valid according to a ledger chain*** *BC if the following properties hold:*

- *For all $0 < i \leq \mathsf{len}(BC), PC^j[i].linkLedger = H(BC[i-1])$*
- $\mathsf{len}(PC^j) - 1 \leq \mathsf{len}(BC)$
- *There exists $k \in BC[0].NA$ such that $k \in PC^j[0].id$*
- *Let k be such that $PC^j[k].link = H(PC^j[0])$, then $j \in BC[k].NA$*
- *For all $i \leq \mathsf{len}(BC)$, if there exists a triple $(x, y, z) \in BC[i].P$ such that $z = \Sigma.\mathsf{sign}^j(x, y)$ then $x = PC^j[i].proof$ and $y = H(PC^j[i])$*

4.1.2 Verification

To be able to participate in the protocol, the participants must verify that the inputs they are receiving are valid in regards to the ledger chain they are maintaining. They must also be able to validate whether the proof-of-work function was computed correctly as well as the validity of signatures.

Proofs of Work We have already presented a way to verify whether a proof of work was computed correctly in the previous chapter. We present the function $\mathsf{verifyPoW}$ again, which takes a triple (x, y, s) as input and verifies whether $y = \mathsf{PoW}^s(x)$. If we want to see whether the proofs of work in a PoW block $PC^j[i]$ are properly computed, we would evaluate $\mathsf{verifyPoW}$ over $(PC^j[i].\mathit{link}, PC^j[i].\mathit{proof})$ (which is a slight abuse of notation, as $PC^j[i].\mathit{proof}$ is a pair). It is important to note that each participant can make unlimited queries to $\mathsf{verifyPoW}$ and that the query takes only one time step and is not dependent on the rate γ_j of the participant.

PoW blocks For a participant a_j to characterise whether a PoW block is valid, he must take into account the structure of the block and whether it is consistent with his ledger chain as well as the PoW chain where it belongs. Checking the structure of the block implies not only checking whether it follows the definition, but also whether $PC^k[i].\mathit{linkLedger}$ is equal to $H(BC[i-1])$. This means that the block points to the appropriate ledger block. Ledger blocks also point to PoW blocks, although not necessarily. If it is the case, then there exists a triple (x, y, z) in $BC[i].P$ such that $z = \Sigma.\mathsf{sign}^k x$ and the following properties must be verified

1. $PC^k[i].\mathit{link}$ is equal to y
2. $PC^k[i].\mathit{proof}$ is equal to x

If this is not the case, a_j must query for $PC^k[i-1]$, verify it and then use $H(PC^k[i-1])$ in place of y (it is not necessary to check the second property as long as the block is well constructed). Note that this procedure works if and only if the preceding chains are consistent and valid. We will call this procedure $\mathsf{verifyPC}$ and any participant may query the environment for it a polynomial amount of time per time step. If the block is successfully verified, $\mathsf{verifyPC}$ outputs 1, in any other case it outputs 0. While in certain cases it is necessary to have the rest of the PoW blockchain to verify a block, for simplicity in the presentation we will only include $PC^k[i]$ and BC as the inputs to $\mathsf{verifyPC}$. Note that $\mathsf{verifyPC}$ makes use of $\mathsf{verifyPoW}$ to check whether the proofs of work are computed correctly.

Ledger Block and Transactions Participants are also interested in verifying blocks from the ledger chain. To verify $BC[i]$, they should check whether it points to the right block in the chain and whether it contains enough correct proofs of work by the participants. Besides the structural properties of the block, we must also ensure that the transactions contained within it are valid. To simplify this, we will define two functions to do this verification. One which only takes into account the consistency of the transactions, $\mathsf{verifyTxs}$ and one which solely checks the structure of the block itself, $\mathsf{verifyBC}$. The former takes a blockchain and a (possibly empty) set of transactions and checks if the transactions in the blockchain are consistent and whether the set can be added to it without breaking consistency. It also checks that all transactions are valid, as only the environment may create valid transactions but the adversary may input fake transactions. The latter takes a blockchain $BC[0,i)$ and a block $BC[i]$, with $i \geq 2$, and checks whether the following properties hold:

- $BC[i].st = i$
- $BC[i-1].NA \subseteq BC[i].NA \cup \mathsf{ind}(BC[i].G)$
- $BC[i-1].NA \setminus BC[i].NA \neq \varnothing$ iff there exist records in $BC[i].G$ such that $\mathsf{ind}(BC[i].G) = BC[i-1].NA \setminus BC[i].NA$
- $BC[i].link$ is equal to $H(BC[i-1])$
- Let $|BC[i].NA| = m$, $|BC[i].P| > Qm$ where every element of P is of the form (x,y,z) with $z = \Sigma.\mathsf{sign}^j(x,y)$.
- $BC[i].G$ contains records of the form (M, B, B^*), where $M \subseteq BC[i-1].NA$ and $M \cap BC[i].NA = \varnothing$
- Let $|BC[i].NA| = m$, $|BC[i].Sig| > Qm$ where every $s \in Sig$ is of the form $s = \Sigma.\mathsf{sign}^j(id, NA, link, T, G, P)$.
- The order of the elements of P and Sig corresponds to the order induced by the indices
- If there exist $(x,y,z) \in BC[i].P$ and $(x^*, y^*, z^*) \in BC[i-1].P$ such that $z = \Sigma.\mathsf{sign}^j(x,y)$ and $z^* = \Sigma.\mathsf{sign}^j(x^*,y^*)$ then $\mathsf{verifyPoW}(y^*, x) = 1$

Note that while verifyTxs checks if the transactions in the chain itself are consistent, verifyBC checks only whether the new block would fit at the tail of the chain, but does not check if the chain itself is constructed correctly. However, this would be easy to do by defining a function that uses verifyBC recursively.

Cheating An important part of our protocol is the ability to recognise and expel cheaters from participating. We are interested in preventing people from forking the chain and, because the blocks are signed, it is possible to infallibly recognise when this happens. If two different blocks with the same round identifier are signed by the same participant, it can only be because of an attempt to fork the chain. Because our signature scheme is assumed to be unforgeable, this can only happen if a party purposefully tried to maintain two different chains[2]. An attempt at forking can be recognised both in ledger and in PoW blocks. We will take advantage of the similarity in their construction to create a function that can work on both types of blocks, which we will call checkCheat.

[2]We will see that it is impossible for an honest party to accidentally sign the incorrect ledger block.

If we have two PoW blocks that belong to the same participant, we have proof that the creator of that block is attempting to fork a chain. However, if we have two ledger blocks that belong to the same round, we only have definite proof of cheating by the people who signed both blocks. Because honest parties will have one of the two blocks in their chains already, it would be tempting to define everyone who signed the blocks from the fork as an antagonistic party and remove them from the protocol. However, we are interested in punishing only overt attempts at cheating: when someone signed two distinct blocks in the same round.

Given two blocks, we want to know if they imply that a certain set of participants were cheating. Although the two block structures that we work with are different, they have the same overall structure. In particular, the last element of the tuple contains signatures of the rest of the block, which is what we care about. This means that given a block B, which might be either a ledger or PoW block, the seventh element, $B.Sig$, is a set of signatures. In the same way, the first element, $B.st$, is a round identifier. Therefore, we define checkCheat the following way:

$$\mathsf{checkCheat}(B, B^*) = \begin{cases} \varnothing & \text{if } B \text{ and } B^* \text{ are of different type} \\ \mathsf{ind}(B.Sig) \cap \mathsf{ind}(B^*.Sig) & \text{if } B.st = B^*.st \\ \varnothing & \text{otherwise} \end{cases}$$

Having a cheating detection mechanism inside of the protocol allows the protocol to regulate itself. While we will work in a permissioned setting where consequences can be executed outside the model, having a mechanism inside of it allows us to regulate behaviour from the inside in order to not rely on outside compliance to keep the protocol functional.

4.2 Consensus

Our blockchain protocol is more focused on the structure of the blockchain itself than on the creation of a consensus mechanism. We will have a subprotocol, named Consensus which will choose and distribute a block each round to all the participants for them to add to their copy of the ledger chain. This subprotocol will itself call another protocol BFT which is the protocol that ensures that all parties agree on something. We use the name BFT because there are Byzantine-Fault-Tolerance protocols that fulfill the properties that we need, for example the one presented in [LVCQ16]. These protocols require a complete knowledge of the network participants and (in most cases) a public-key infrastructure. However, any protocol that can ensure the same characteristics may be substituted in here.

Following the universal composability framework, we will treat BFT as an ideal functionality that is called by each participant by inputting the indices of the other participants who will be running the protocol, *comm*, as well as a set containing a log that the parties have agreed on beforehand, *hist*. At initialisation, each participant inputs a set of elements I from which the output will be constructed. At the end of the execution, BFT will output a set constructed from the inputs of the parties. If enough (more than a fraction Q of the total) participants are honest, BFT will output the same set to all participants. The participants running this protocol have access to every aspect of the public-key functionality. The protocol BFT has access to verification functions over the inputs (to check that they are valid) and the output (to ensure that it fulfills expected properties), we will elaborate more on this topic later. It works in the following way:

Inputs Each participant a_j in BFT is allowed to input only the following:

- The initialisation input `start`$(comm, hist, I)$
- The input `halt` stops the execution of the protocol by a_j

Outputs

- After a successful execution (one without a `halt` input) BFT outputs an ordered list of records O.

To characterise a correct execution of this protocol, we define the liveness parameter $L_{\mathsf{BFT}} \in \mathbb{N}$, which is a function of the number of participants n. We also define the network delay $\delta \geq 0$, which represents the number of time steps that $\mathcal{A}$ is allowed to delay a message by an honest participant. A **compliant execution of** BFT by the pair $(\mathcal{Z}, \mathcal{A})$ given n and L_{BFT} fulfills the following properties:

Bounded Network Delay Every message output by an honest participant reaches every other honest participant at most δ time steps after it was sent.

Initialisation Agreement If an honest party receives input `start`$(comm, hist, I)$, then every honest party in the execution that received a `start` input, received it with the same *hist* and *comm* and with a different *sk*.

Static Corruption Given that the participants in *comm* are running the protocol, no more than $(1-Q)|comm|$ of the participants are controlled by $\mathcal{A}$ at the beginning of the execution and no participants become corrupted during the execution of BFT.

Close Start and Stop If an honest party receives an input `start`$(comm, hist, I)$ or `halt` before any other honest party, all honest parties in *comm* will receive it within δ time steps.

An important property encoded in the system is the idea of a *consistent* record. Participants in our protocol will call BFT whenever they need to agree on something. Each instance of BFT depends on the type of information the participants are agreeing on, and therefore entails different rules. An adversarial party may try to input records that are not valid. The participants in the protocol have a way of recognising these records and preventing them from being added to the output. In our protocol we are interested in using it to agree on many different things, so we will differentiate these instances by a superscript. When the participants want to agree on the ordering of transactions, we will write BFT^T. Transactions have specific rules that must be fulfilled to create a valid ordering of them, encoded within $\mathsf{verifyTxs}$. When we use BFT for proofs of work (PoW^P) or signatures (PoW^S), the necessary conditions are different ($\mathsf{verifyPoW}$ and $\{\Sigma.\mathsf{verify}^k \mid k \in N\}$ respectively). In the cases where it is necessary to have information beyond what is added as an input (like in the case of transactions), we will use the variable *hist*. We will speak of **consistent** outputs when they fulfill the requisite properties. We will also speak of **maximally consistent** outputs when no element in any participant's input can be added without violating these restrictions.[3]

We want the BFT protocol to be secure. In this case security means that the adversary cannot make an honest party accept a different ledger and that it cannot stop a particular record from being added to the ledger of accepted records. This is captured by the following:

Consistency Whenever an honest node outputs a set *log* after being given *hist* as an input, $hist + log$ is consistent, where $+$ denotes concatenation.

Liveness At most L_{BFT} time steps after receiving a `start` input, BFT will end and output a maximally consistent subset of the union of all inputs.

Agreement When BFT outputs a set O to an honest participant, it will output the same set O to all other honest participants within δ time steps. If an honest participant inputs `halt` to BFT after another honest party has received O, they will receive O regardless of the `halt` input.

BFT is a straightforward protocol which does not require interaction from the environment and outputs a single result. We can look at it as a non-deterministic function from the inputs in the `start` command to the output generated at the end of the execution. We will introduce an abuse of notation and say that $\mathsf{BFT}(comm, hist, I)$ is equal to the final output of an execution of BFT with initialisation input $\mathtt{start}(comm, hist, I)$ which successfully terminated. The fact that we will use BFT every round might seem insecure due to the result in [LLR06] which shows that deterministic byzantine-fault-tolerance protocols cannot be sequentially composed indefinitely. However, this is not a problem in our case, as that weakness depends on the fact that it is impossible for parties to recognise to what instance of the protocol a message belongs to. However, they present a workaround that permits unlimited instances based on session

[3]Note that due to our restrictions on transactions, all transactions will be consistent with each other, so the maximally consistent set will be all of the transactions

identifiers. While this is generally problematic, as the common information must come from an external source, information from the previous ledger block will act as the identifier.

4.2.1 The Consensus subprotocol

The protocol BFT, while powerful, does not capture all of the properties we want from the process of choosing a new block. It does not include the process of ejecting antagonistic participants from the protocol or how participants that do not take part in consensus get a block. In particular, it does not take into account the process of signing the block. For that, we will define another protocol that extends BFT in the following ways:

Participation of non-committee members In this simple model, every participant should be participating in the BFT instance if they are following the protocol. However, as we relax our network assumption (for example, by permitting late spawning) we will have a way for new participants to start contributing proofs of work and signatures before they are able to participate in the process of choosing the transactions in a block.

Termination Whenever this protocol is run, it will not stop until it gets an input to halt. Whenever it gets an input `halt`, it will run all subprotocols until termination and then stop, outputting the same block to all the honest participants. This implies that termination is not immediate after receiving an input, receiving an input to halt simply entails that the protocol can terminate and that the protocol will stop whenever the subprotocols reach a successful execution .

Antagonistic Elimination If a proof of cheating is input to the protocol, the antagonistic parties will be expelled from the protocol and will be unable to continue to influence the protocol. This event will not interfere with the execution of the protocol but the consequences of it will be reflected on the output.

The interaction of the environment with this subprotocol is defined by the following:

Inputs

- The initialisation input `start`$(BC, comm, W, TxPool, G)$
- The input `halt` stops the execution of the protocol by a_j
- The input `cheat`(B, B^*) which contains two distinct blocks from the same round that were signed by the same participant

Outputs

- After a successful execution, Consensus outputs a signed block $(block, S^*)$

$\mathsf{Consensus}$ will make extensive use of the BFT protocol for agreement and will be run by a subset of all participants. In the models in this thesis, this will consist of all non-antagonistic participants, but the conditions can be modified so that only a subset of the participants will run it. We will see more about this in the next chapter. Using BFT is necessary, as there must be total agreement on the content of the block due to the fact that the block includes contributions (proofs of work and signatures) from everyone. Therefore, there is a minimum time necessary for this protocol to function correctly, based on the time it takes to run BFT enough times to be in total agreement on transactions and signatures. We will call the upper bound for this time interval $L_{\mathsf{Consensus}}$. This protocol also has access to all the verification functions that we defined previously. While we assume that these functions are the ones that we mentioned, these can be redefined in case they are attempted to be used in a different setting.

Given the protocol, we must characterise a **compliant execution of** $\mathsf{Consensus}$. Given an environment-adversary pair $(\mathcal{Z}, \mathcal{A})$, we say the pair is $(n, Q, L_{\mathsf{BFT}}, \delta)$**-valid** w.r.t. $\mathsf{Consensus}$ if the following properties hold:

Bounded Network Delay Every message output by an honest participant reaches every other honest participant at most δ time steps after it was sent.

Initialisation Agreement Whenever an honest party is initialised by an input $\texttt{start}(BC, comm, W, TxPool)$, every honest party in the execution that received a `start` input, received it with the same BC, $comm$, W.

Valid Initialisation When an honest party receives the initialisation input $\texttt{start}(BC, comm, W, TxPool)$, BC must be a valid ledger chain, W must have at least $|\mathsf{last}(BC).NA|Q+1$ blocks B^j of which at least Q-percentage of them must come from honest players and have the property that if $\big(x, y, \Sigma.\mathsf{sign}^j(x, y)\big) \in \mathsf{last}(BC).P$ then $\mathsf{verifyPoW}(y, B^j.\mathit{proof}) = 1$. The set $TxPool$ must be a subset of $\mathfrak{T}$ and include only valid transactions.

Static Corruption Given that $comm$ participants are running the protocol, no more than $(1 - Q)|comm|$ of the participants who received a `start` command are controlled by $\mathcal{A}$ at the beginning of the execution and no participants become corrupted during the execution of $\mathsf{Consensus}$.

Close Start and Stop If an honest party receives an input `start` or `halt` before any other honest party, all honest parties in $comm$ will receive it within δ time steps.

Cheating Timeliness Whenever a `cheat` input is received by an honest participant, it is received by all honest participants. Furthermore, no `cheat` inputs are received after receiving a `halt` input by honest parties. Note that we cannot prevent the adversary from inputting `cheat` at inopportune times. However, this will not affect the execution of the protocol, as honest parties will ignore this input when it arrives too late.

```
Consensus for participant $a_j$

On input start($BC, comm, W, TxPool$):
  • Set $Ni = \mathsf{last}(BC).NA \cup \mathsf{ind}(W)$ and $G = \varnothing$
  • Set $W^c = \{\mathsf{formatPoW}(x) \mid x \in W, \mathsf{verifyPC}(x, BC) = 1\}$
  • Set $comm^{Txs} = \mathsf{ind}(W^C) \cap comm$

  If $\mathbf{j} \in \mathbf{comm}^{Txs}$ and $|\mathbf{comm}^{Txs}| \geq \mathbf{Q}|\boldsymbol{comm}|$
    • Call $\mathsf{BFT}^T$ with input start($comm^{Txs}, BC.T, TxPool$)
    • Let $T$ be the output of $\mathsf{BFT}^T$
    • Broadcast $T$

  Else
    Wait until $(1-Q)|comm|+1$ participants send the same $T$, save $T$

  On input halt: Wait until $\mathsf{BFT}^T$ successfully terminates
  • Let $block = \big(\mathsf{len}(BC)+1, Ni, H\big(\mathsf{last}(BC)\big), T, G, W^c\big)$
  • Broadcast $block$
  • Call $\mathsf{BFT}^S$ with input start$\big(comm, \varnothing, \{\Sigma.\mathsf{sign}^j(block)\}\big)$
  • Let $S^*$ be the output of $\mathsf{BFT}^S$
  • Output $(block, S^*)$ and halt

On input cheat($B, B^*$): Set $Ch = \mathsf{checkCheat}(B, B^*)$
  If $\boldsymbol{Ch} \cap comm \neq \varnothing$
    • Update $comm = comm \setminus Ch$ and $Ni = Ni \setminus Ch$
    • Update $comm^{Txs} = comm^{Txs} \setminus Ch$
    • Eliminate contributions from participants in $Ch$ from $W^c$ and
      ignore communications from them
    • Add $(Ch, B, B^*)$ to $G$

    If running $\mathsf{BFT}^T$ and halt has not been input
      • Input halt to $\mathsf{BFT}^T$
      • Call $\mathsf{BFT}^T$ with input start($comm^{Txs}, BC.T, TxPool, sk$)
```

The function $\mathsf{formatPoW}(x)$ is a function that takes a PoW block and encodes it in the way that it will be represented in the blockchain. That is, as a triple including the proof itself, a hash of the block and the signature of the block. Note a simple abuse in notation, where $(block, S^*)$ represents a fully formed ledger block, and not a pair between an unsigned block and its signatures.

It is very important that the signatures are agreed upon perfectly, as everything else in the block. If someone does not have the same signatures as everyone else, their block will be different and therefore their PoW chain will be too. We need to ensure *agreement* between all participants. That is why we have to make sure that every participant receives the same set of signatures, with no exceptions. Therefore, we use BFT to agree over the signatures. Because the adversary is in charge of delivering the messages, she has a lot of power over this process and can easily make it fail even when we add waiting times to each communication round. Instead of trying to find a particular solution for this issue, we instead use something that we already have access to: BFT. it is important to note that the implementation of BFT^S does not need to be the same as BFT^T. Due to the difference in the nature of the data that we are agreeing upon, it would make sense to have different implementations of consensus for each of these instances.

We define security in the Consensus protocol as follows:

Definition 4.5. *An execution of* Consensus *is said to be secure against* $(1-Q)$-*corruption with liveness parameter* L_{BFT} *if for any* $n > 0$ *and for any pair* $(\mathcal{Z}, \mathcal{A})$ *that is* $(n, Q, L_{\mathsf{BFT}}, \delta)$***-valid*** *w.r.t.* Consensus*, there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$*, except with* $\mathsf{negl}(\lambda)$ *probability, the following properties hold for* $\text{EXEC}_{(\mathcal{Z},\mathcal{A})}[\mathsf{Consensus}]$*:*

Structural Consistency *: For any honest participant, if they input a valid BC in the* `start` *command, the chain resulting from concatenating the output* $(block, S)$ *at the end of BC is valid w.r.t.* verifyBC*.*

Content Consistency *: For any honest participant, if they input a valid BC in the* `start` *command, the transaction contained in the output are valid with the ones in BC w.r.t.* verifyTxs*.*

Liveness *: If a valid transaction tx is contained in TxPool in the* `start` *input of an honest party, then* $tx \in block.T$

Agreement *When an honest party receives* $(block, S)$ *as an output from* Consensus*, every other honest participant receives the same* $(block, S)$*.*

Given this construction of the protocol, we can prove that Consensus is secure and therefore does what we expect of it

Lemma 4.6. *Given a compliant execution of* Consensus*, every execution of* BFT *by* Consensus *is compliant.*

Proof. We need to fulfill the following properties:

Bounded Network Delay By *Bounded Network Delay* of Consensus

Initialisation Agreement If an honest party receives input $\texttt{start}(comm, hist, I)$, then every honest party in the execution that received a $\texttt{start}$ input, received it with the same *hist* and *comm*. Whenever an honest party receives an initialisation input, it contains either an empty *hist* in the case of BFT^S or the transactions in BC for BFT^T. Due to Consensus's *Initialisation Agreement*, we know that BC is the same for all participants. $comm^{Txs}$ is the same for all participants, as it is based over W and BC (same by *Initialisation Agreement* of Consensus) and modified only by the cheat command, which is broadcast and verifiable by everyone.

Static Corruption By *Valid Initialisation* of Consensus, W must contain a Q-majority of valid PoW blocks from honest parties which appear in the previous block. All of the parties that contributed a valid PoW block will be in $comm^{Txs}$ by construction, an no one else. Thus, the committee for BFT will have a Q-majority.

Minimum Run By construction, Consensus will always wait for BFT^T and BFT^S to finish running.

Close Start and Stop Follows from Consensus's *Close Start and Stop*, *Bounded Network Delay* and *Cheating Timeliness.*

□

Theorem 4.7 (Consensus from BFT)**.** *Suppose that the signature scheme is secure, that H and* PoW *are independent random oracles parametrised by λ. Suppose that* BFT *is secure against $(1-Q)$-corruption with liveness parameter L_{BFT} for $Q > 1/2$. Then* Consensus *is secure against $(1-Q)$-corruption given a pair $(\mathcal{Z}, \mathcal{A})$ that is $(n, Q, L_{\mathsf{BFT}}, \delta)$-**valid** w.r.t.* Consensus*.*

Proof. To ensure security, we must show that the following properties hold:

Structural Consistency : We must ensure that verifyBC holds for $BC + (block, S)$, however, we only care about the last block (which we will refer to as $BC[i]$ for clarity) as we assume that the rest of BC is valid due to *Valid Initialisation.* We need the following properties to hold, given that $|BC[i].NA| = m$:

- $\mathbf{BC[i].st = i}$: By construction
- $\mathbf{BC[i]}.NA \subseteq \mathbf{BC[i-1]}.NA$: $BC[i-1].NA$ was input to Consensus and elements can only be eliminated from it, not added, before it gets added to $BC[i]$
- $\mathbf{BC[i].link}$ is equal to $H(BC[i-1])$: By construction
- $|\mathbf{BC[i].P}| > \mathbf{Qm}$: Follows from *Valid Initialisation*
- G consists of outputs from checkCheat, which have the form that we expect

- $|\mathbf{BC[i].Sigs}| > \mathbf{Qm}$: Because every honest party follows the protocol and due to the *Agreement* property of BFT, every honest party agrees on the same *block* and therefore their signature is valid and added to the block. As we saw previously, our bounds on communication also ensure that there is agreement over the signatures of the block. By *Static Corruption*, there will be at least Qm participants to sign the block.
- **Ordering of P and $Sigs$**: By construction
- If there exist $(x, y, z) \in BC[i].P$ and $(x^*, y^*, z^*) \in BC[i-1].P$ such that $z = \mathsf{sign}^j(x, y)$ and $z^* = \mathsf{sign}^j(x^*, y^*)$ then $\mathsf{verifyPoW}(\mathbf{y^*}, \mathbf{x}) = \mathbf{1}$: Follows from *Valid Initialisation.*

Content Consistency : Follows directly from consistency of BFT

Liveness : If a valid transaction tx is contained in $TxPool$ in the `start` input of an honest party, then it will be input into every instance of BFT^T run in that round. Eventually, one instance of BFT^T will run for L_{BFT} steps and terminate. Therefore BFT^T will output a maximally consistent subset of the union of the transactions. because all transactions that were input are assumed to be consistent, tx will be in the output of BFT^T

Agreement : The st and $link$ come directly from BC, which are the same by *Committee Agreement.* T, P and $Sigs$ are consequences of running BFT, and therefore inherit *Agreement* from there. NA comes from BC but can change because of cheating. Similarly, G is empty unless there is cheating. By construction, whenever an honest party changes any of them, every honest party will change both.

□

4.3 The Idealised Broadcast Protocol

Having defined the subprotocol to agree on each block, we now turn our attention to the main protocol. The principal goal of this main protocol, $\mathsf{IdealLipwig}^\tau$, is to maintain the blockchains and collect proofs of work. Most of the work of the protocol happens in $\mathsf{Consensus}$ but $\mathsf{IdealLipwig}^\tau$ is in charge of maintaining the desirable context over which we will want to run $\mathsf{Consensus}$. The protocol extends $\mathsf{Consensus}$ in the following ways:

Continuous Execution While the previous protocols we have explored in this chapter reach an output and stop, the $\mathsf{IdealLipwig}^\tau$ protocol is always being run, maintaining the blockchains in the process. As such, it has an initialisation procedure and runs continuously, without running towards an output.

Adaptive Corruption In other protocols we assumed static corruption. As the protocols run only within one round of $\mathsf{IdealLipwig}^\tau$, it is reasonable to assume that it would not be enough time for an adversary to corrupt them. However, within $\mathsf{IdealLipwig}^{\tau}$ we will include adaptive corruption. The adversary $\mathcal{A}$ can send a message `corrupt`(a_j) to $\mathcal{Z}$ at any time. If $\mathcal{A}$ has not reached its threshold of corrupted participant, it will gain control of a_j at the beginning of the next round. This permits us to assume that every subprotocol only has to protect against static corruptions. In practice if a participant is corrupted at any point within a round, we treat it as if he had been corrupted at the beginning of the round.

Transaction Introduction The protocol $\mathsf{IdealLipwig}^{\tau}$ is connected to a transaction pool $\mathsf{TxsPool}$. At every time step, $\mathcal{Z}$ adds valid transactions to Txs. At any time, any participant may query $\mathsf{TxsPool}$ for transactions to input into $\mathsf{Consensus}$. At the end of each round, each participant inputs the most recent ledger block to $\mathsf{TxsPool}$, which erases every transaction registered in that block from it.

Connection with Golem In this protocol, each participant can run the protocol Golem_{γ} defined in the previous chapter. Because we assume every participant has the same rate, we will avoid using the subscript for the rate and will refer to the rate of every player as γ in all this chapter.

We will make a very important simplification in this protocol, and that is that communication is immediate, that is $\delta = 0$. While both BFT and $\mathsf{Consensus}$ are more robust and can still work given a network delay, this model will assume there is no delay and communication is immediate (that is, at the end of each time step, each participant receives every message sent during that time step).

The protocol $\mathsf{IdealLipwig}^{\tau}$ works as follows: At the beginning of the protocol $\mathcal{Z}$ initialises n parties, the adversary might choose to corrupt any number of them (within its limits) at that time or wait to corrupt them later. Then, $\mathcal{Z}$ randomly generates private and secret keys pk_j and sk_j for each of them and stores them. It outputs $BC[0], PC^j[0], \mathsf{PoW}(PC^j[0])$ as well as all the relevant keys to each participant a_j. The protocol for participant a_j is defined as follows:

Ideal Broadcast Protocol $\mathsf{IdealLipwig}^{\tau}$ for a_j

Initialisation

Upon activation by $\mathcal{Z}$

- Receive (pk_j, sk_j) and the list of pairs (k, pk_k) from $\mathcal{Z}$
- Receive $BC[0], PC^j[0]$ from $\mathcal{Z}$
- Receive $\big(\mathsf{PoW}^{\tau\gamma}(PC^j[0]), \tau\gamma\big)$ from $\mathcal{Z}$ and immediately start Round 1

Round i

The round begins upon reception of the output of Golem_γ:
$G_i = \big(\mathsf{PoW}^s(PC^j[i-1]), s\big)$

PoW Chain Maintenance:

- Set $\sigma = \mathsf{sign}^j\big(H(PC^j[i-1]), H(BC[i-1]), \perp, G_i\big)$
- Append $\big(H(PC^j[i{-}1]), H(BC[i{-}1]), \perp, G_i, \sigma\big)$ to PC^j as $PC^j[i]$
- Query for $\mathsf{Golem}_\gamma(PC^j[i])$

Pre-Consensus**:**

- Let $comm = \mathsf{last}(BC).NA$
- Broadcast $PC^j[i]$
- Let P be the set of every *PoW* block received
- Broadcast P
- Upon reception of the *PoW* block sets, take the majority P' and count how many participants sent it.
  - **If** less than $(1 - Q/2)|comm| + 1$ participants agree on P', broadcast `runBFT` and in the next time step, set $P^* = \mathsf{BFT}^P(comm, \varnothing, P)$.
  - Otherwise, wait one time step. **If** no `runBFT` message is received set $P^* = P'$. Otherwise , set $P^* = \mathsf{BFT}^P(comm, \varnothing, P)$
- Input `receive` to TxsPool, get back Txs
- **Call** Consensus(BC, $comm$, P^*, Txs, $\perp$)
- After $\tau - (2L_{\mathsf{BFT}} + 5)$ time steps, input `halt` to Consensus
- Wait until Consensus *outputs* a block, add it to BC
- Input `clean` $BC[i].T$ to TxsPool

- After τ time steps since the beginning of the round, **input** `halt` to Golem_γ

Cheating Recognition

Whenever a_j receives a signed block B^* that is different from the one in the same stage B

- **Broadcast** (B, B^*)
- **If** Consensus is running and `halt` has not be input: input `cheat`(B, B^*) to Consensus
  Otherwise: Wait until the next instance of Consensus is running, then input `cheat`(B, B^*)

On **input** `chain`: Output BC
On **input** `PoWchain`: Output PC^j

When agreeing over the proofs of work, we have a similar situation that we had in Consensus to agree over the signatures. One would think that because $\mathsf{IdealLipwig}^{\tau}$ assumes a perfectly synchronous network it can be possible to agree on proofs without resorting to BFT, considering that the adversary has considerably less power to disrupt the protocol. However, due to the adversary's ability to send different messages to different participants, she can still affect this process. Our solution does not manage to completely solve this. While our bound prevents the adversary from interrupting the protocol, we cannot prevent her from forcing a call to an instance of BFT^{P}. This is unfortunate, as we must take into account a complete run of BFT (taking L_{BFT} time steps) when we set the length of the rounds, τ, even if we could avoid running BFT^{T} in some rounds of the protocol. Note that we are not constrained to always use BFT to agree on the proofs of work, as the order of the proofs is determined by the ordering on the indices. It is notable that the threshold for ensuring agreement is higher than having agreement between all honest parties. This is due to an attack that the adversary can employ to split the honest parties into two groups that hold different sets but believe that $(1 - Q/2)|comm|$ other parties have the same set.

Lemma 4.8. *Given an execution of* $\mathsf{IdealLipwig}^{\tau}$ *with honest percentage* Q*,* n *participants and adversary* $\mathcal{A}$*, if honest parties accept less than* $(1 - Q/2)n + 1$ *equal sets of proofs of work when creating* P^**,* $\mathcal{A}$ *can split the honest parties in two groups such that each group has a different* P^* *and does not realise it.*

Proof. We will present an attack in which $\mathcal{A}$ splits the honest parties in such a way that every honest participant receives one of two distinct candidate sets (P_1 or P_2) from $(1-Q/2)n$ different participants. Given n participants in the protocol and an honest percentage of Q, the adversary divides the honest parties into two sets (H_1 and H_2) with $Qn/2$ participants each and divides the adversarial participants into other two sets (A_1 and A_2) with $(1-Q)n/2$ participants each. When sending proofs of work, A_1 sends PoWs only to H_1 and A_2 sends them only to H_2. In the next step, all adversarial parties send the PoWs from H_1, A_1 and A_2 (a total of $Qn/2 + 2(1-Q)n/2 = (1 - Q/2)n$) to H_1 and the PoWs from H_2, A_1 and A_2 to H_2. This means that the honest parties will have distinct sets with $(1 - Q/2)n$ PoWs. □

Given this result, we know that the adversary cannot convince every honest participant that everyone else got the correct P^*. However, she can still convince one that this is the case to force the execution of BFT^{T}. She does not need to do this, as simply sending a message `runBFT` will do the trick.

Given the protocol, we must characterise a **compliant execution of** $\mathsf{IdealLipwig}^{\tau}$. Given an environment-adversary pair $(\mathcal{Z}, \mathcal{A})$, we say the pair is $(n, Q, L_{\mathsf{BFT}}, \delta)$**-valid** w.r.t. $\mathsf{IdealLipwig}^{\tau}$ if the following properties hold:

Ideal Network Properties Every message output by an honest participant reaches every other honest participant by the end of the time step in which it was sent. That is, $\delta = 0$

Simultaneous Initialisation All n parties are initialised at the same time at the beginning of the protocol. Before they are initialised, the adversary might communicate to $\mathcal{Z}$ which parties it wishes to corrupt. These parties will start the protocol already corrupted, as long as they constitute no more than $(1-Q)n$ participants.

Valid Initialisation At the beginning of the protocol, every participant receives the same correctly constructed $BC[0]$ and also receives valid $PC^j[0]$, $\big(\mathsf{PoW}^{\tau\gamma}(PC^j[0]), \tau\gamma\big)$ and a pair of keys (pk_j, sk_j) that are shared by no other participant. They also receive the correct list of association between indices and public keys.

Adaptive Corruption At any round $\mathcal{A}$ might send a message $\texttt{corrupt}(a_j)$ to $\mathcal{Z}$. At the beginning of the next round, $\mathcal{A}$ will gain control of a_j as long as that means that no more than $(1-Q)n$ participants are being controlled by $\mathcal{A}$.

Timeliness Every participant queries Golem_γ at the same time

Minimum Run We have that $\tau\, L_{\mathsf{Consensus}} + 1 = O(2L_{\mathsf{BFT}})$

We assume that all non-adversarial parties follow the protocol to the letter. Because we assume at least Q of the participants are honest, this entails that $\mathsf{verifyPoW}(BC, PC^k[i])$ will always be true for all honest a_k. Therefore, at least Q of the participants of $\mathsf{Consensus}$ will be honest, which means that it will work as expected. Therefore, a valid (because of *consistency* in $\mathsf{Consensus}$) block will be issued at every round and, due to *correctness* of $\mathsf{Consensus}$, will be adopted by all honest parties. More formally, we define security in the $\mathsf{IdealLipwig}^\tau$ protocol as follows:

Definition 4.9. *Given random oracles H and PoW parametrised by λ, $\mathsf{IdealLipwig}^\tau$ is said to be secure against $(1-Q)$-corruption with liveness parameter L_{BFT} if for any $n > 0$ and for any pair $(\mathcal{Z}, \mathcal{A})$ that is $(n, Q, L_{\mathsf{BFT}}, Q, \delta)$-**valid** w.r.t. $\mathsf{IdealLipwig}^\tau$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, except with $\mathsf{negl}(\lambda)$ probability, the following properties hold for $\mathrm{EXEC}_{(\mathcal{Z},\mathcal{A})}[\mathsf{IdealLipwig}^\tau]$:*

Consistency *: If an honest participant a_j is queried* $\texttt{chain}$ *at time t and outputs BC and an honest participant a_k (possibly equal to a_j) is queried* $\texttt{chain}$ *at time $t^* \geq t$ and outputs BC^*, we have that $BC \prec BC^*$. If $t = t^*$ we have that $BC = BC^*$*

Liveness *: If a valid transaction tx appears in* $\mathsf{TxsPool}$ *in round i then either $tx \in BC[i]$ or $tx \in BC[i+1]$*

Lemma 4.10. *Given a compliant execution of* $\mathsf{IdealLipwig}^\tau$*, every execution of* $\mathsf{Consensus}$ *by* $\mathsf{IdealLipwig}^\tau$ *is compliant.*

Proof. We must prove the following hold

Bounded Network Delay Follows from $\mathsf{IdealLipwig}^{\mathcal{T}}$'s *Ideal Network Properties*

Initialisation Agreement Consensus is initialised with the following variables that must match:

- **BC**: Every participant has the same genesis block from *Valid Initialisation* of $\mathsf{IdealLipwig}^{\mathcal{T}}$. So in Round 1, this holds. Every other block is generated by Consensus. Because of Consensus's *Agreement*, by induction, every honest participant always holds the same BC.
- *comm*: As seen in the previous point, every honest participant holds the same BC, so $\mathsf{last}(BC).NA$ is always the same. This implies *comm* has the same initial value. However, this can change when cheating is encountered, but because everyone becomes aware of cheating, it will be changed accordingly by everyone.
- **W**: W depends on BC, P^* and verifyPC, all of which are equal for everyone (BC by Consensus's *Agreement* and P^* by BFT's *Agreement*)
- R: All participants receive an empty string $\perp$

Valid Initialisation By construction. There are enough proofs of work in W because honest participants always create a proof of work.

Static Corruption Follows directly from the bounds on adversarial power.

Close Start and Stop By construction, follows immediately.

Cheating Timeliness By construction, if the announcement of cheating would be too late, honest participants wait one round before inputting `cheat`.

□

Lemma 4.11. *Given a compliant execution of* $\mathsf{IdealLipwig}^{\mathcal{T}}$*, every execution of* BFT *by* $\mathsf{IdealLipwig}^{\mathcal{T}}$ *is compliant*

This proof follows a very similar route to Lemma 4.6, so we refrain from writing it. The only important point to note is the time step skipped when running BFT^{T} in order to fulfill *Close Start and Stop* for BFT.

Theorem 4.12 ($\mathsf{IdealLipwig}^{\mathcal{T}}$ from Consensus and BFT)**.** *Suppose that the signature scheme is secure, that H and* PoW *are independent random oracles parametrised by λ. Suppose that* Consensus *and* BFT *are secure against* $(1-Q)$*-corruption with liveness parameter* L_{BFT} *for* $Q > 1/2$*. Then* $\mathsf{IdealLipwig}^{\mathcal{T}}$ *is secure against* $(1-Q)$*-corruption given a pair* $(\mathcal{Z}, \mathcal{A})$ *that is* $(n, Q, L_{\mathsf{BFT}}, \delta)$***-valid*** *w.r.t.* $\mathsf{IdealLipwig}^{\mathcal{T}}$.

Proof. **Consistency** : Whenever any participant adds a block to the chain, it comes from Consensus. By the *Agreement* property of Consensus, this will also be the same. By construction of $\mathsf{IdealLipwig}^{\mathcal{T}}$, every participant will get a new block at the same time.

Liveness : Suppose a valid transaction tx appears in TxsPool in round i, there are two cases:

- tx **appears before querying** TxsPool: tx is contained in the Txs pool that is input into Consensus. *Liveness* in Consensus implies that tx will be contained in the block issued at the end of the execution of the subprotocol.
- tx **appears after querying** TxsPool: tx remains in TxsPool and is not eliminated when the participant inputs the accepted block. Therefore, in the next round, when TxsPool is queried, tx will be in the set it outputs. Then we follow the previous proof to show that $tx \in BC[i+1]$

□

4.4 Immutability

In this chapter we have constructed a model of a blockchain that uses serial proofs of work. While we use these proofs of work to ensure that the participants are investing work into maintaining the blockchain, they do not take part in the process of choosing a new block. The primary purpose of the serial proofs of work is to ensure immutability of the chain. Even in a permissioned network, where there is a certain baseline for trust, a big advantage of blockchains is the ability to transfer trust from the participants to cryptography. Because cryptography is not subject to outside pressures in the same way people are, the ability to shift trust away from agents is very useful. Serial proofs of work ensure that a certain amount of work has to be performed by the participants if they want to modify the blockchain in any way. More importantly, it requires an investment of *time*, as it is impossible to parallelise the work needed to create a proof of work. This is the main advantage over using Nakamoto proofs of work, especially in a permissioned network.

Besides their use as in cryptocurrencies, blockchains show potential to be applied in other aspects. In particular, blockchains are being suggested as a way for different companies to cooperate, as they do not need to trust each other to know that previous information was not modified, which would be the case with traditional ledgers. This use case would require a permissioned blockchain, like the one we are building here. Due to the fact that permissioned blockchains have a more advantageous setting, we do not need protocols that are as robust as the permissionless setting. This permits the use of protocols which use byzantine-fault tolerance to agree on blocks. However, there have been criticisms of these implementations. Agents running the protocol in a permissioned setting will likely run the same software, not only for the protocol itself but also the same operationg system, meaning that there is a practical possibility that the whole network might fall under the control of the adversary. In this case, an adversary would be able to create an arbitrary number of valid blockchains that will be indistinguishable from the real blockchain [Sir17]. Therefore, the immutability claims disappear. Proofs of work prevent these situations from happening, as the adversary cannot create new chains without investing the requisite computational power and time.

In our protocol, each participant uses one single core to compute the serial proof of work. Because they are serial, running them in multiple cores is useless. Because they build on each other, it is impossible to start computing one before all the previous ones have finished computing. Because they are signed by each participant, there is no incentive for anyone to compute more than one. Therefore, the only leverage an adversary can have is access to a faster core that permits them to take less time to calculate one. Even then, they cannot pre-compute future proofs of work because they need to know the ledger block to have a PoW block over which they will apply PoW. This shows that we do not fall into the same pitfalls as the Nakamoto proofs of work, but can we ensure immutability with these new proofs of work?

In the previous chapter we have already shown that PoW chains achieve immutability through the serial proofs of work. We are then interested in showing that this immutability can be *borrowed* by the ledger chain. While the ledger chain contains all the proofs of work computed by the participants for their PoW chains, these are not computed over the blocks of the ledger chain, but over the PoW blocks. This means that the security guarantees that we have on the PoW chains do not translate directly to the ledger chain. In particular, the ledger chain alone cannot prove its own immutability, as it is necessary to have access to at least one linked PoW chain. While this is not ideal, it is also not too much of an issue, as every participant will maintain their own PoW chain.

An important thing to note is that the process in which a participant maintains their PoW chain in $\mathsf{IdealLipwig}^{\tau}$ is the same as the one he would follow in $\mathsf{SingleLipwig}^{\tau}$, but instead that the component *linkLedger* of the block includes the hash of the current ledger block instead of an arbitrary string. In essence, the protocol $\mathsf{SingleLipwig}^{\tau}$ is contained in $\mathsf{IdealLipwig}^{\tau}$. Therefore, we can import all the theorems we have for the immutability of PoW chains into this new setting. This is not enough to show the immutability of the ledger chain, but we will show that it inherits the immutability guarantees from the PoW chains it is connected to.

Before doing that, we will see that no participant in $\mathsf{IdealLipwig}^{\tau}$ can be fooled by a rewritten chain. Every player maintains their own PoW chain, which has a record of how the blockchain looked when each block was created in the form of *linkLedger*. Therefore, any honest party will immediately recognise a rewritten ledger chain.

Lemma 4.13. *In protocol* $\mathsf{IdealLipwig}^{\tau}$*, regardless of the percentage of participants controlled by the adversary, any attempt at replacing blocks in the ledger* *chain can be recognised by any honest party a_j.*

Proof. Suppose that at round $i + r$ with $r > 0$, a block is presented in the consensus stage that points to a ledger chain BC^* that differs from the one held by a_j starting from block i. For $BC^*[i, i+r)$ to be considered valid, each block on it must have at least nQ valid signatures. Therefore, for every $k \in \{i, \dots, i-r+1\}$ we have that $BC[k].Sigs \cap BC^*[k].Sigs \neq \varnothing$. Therefore, each honest party a_j can add all participants represented in $\bigcup_{k\in\{i-r,\dots,i-1\}} BC[k].Sigs \cap BC^*[k].Sigs$ to $BC[i].G$ using the cheating recognition process and continue the protocol with an unforgeable proof of the malfeasance of the participants. □

Therefore, an honest party will never accept a rewritten blockchain unless the adversary managed to erase his PoW chain and replace it with a new one. However, this will require the adversary to have access to the participant's private key, which it can only manage by corrupting said participant. More importantly, replacing a PoW chain requires time, as seen in Theorem 3.10. In order to fool an honest participant, the adversary would have to rewrite his PoW chain. As we have seen in the previous chapter, the problems with doing this extend beyond acquiring access to the relevant secret keys. First, we will prove that ledger blocks are time-locked through the hash pointers between them and the PoW blocks.

Theorem 4.14. *Let BC be a ledger chain created in $\mathsf{IdealLipwig}^\tau$ with H and PoW parametrised by λ and rate γ such that $\mathsf{len}(PC) = i + r$. Any block $BC[i]$ is time-locked by*

$$(r-1)(\tau\gamma/\gamma^* + 1) + 1$$

if γ^* is the upper bound for the rate, except with negligible probability in the security parameter λ.

Proof. Considering that for every $j \in \{1, 2, \dots, n\}$ every PoW block $PC^j[i+1]$ has a pointer to $BC[i]$ and those blocks are time locked by $(r-1)(\tau\gamma/\gamma^* + 1)$ by Theorem 3.5, $BC[i]$ must have been created at least one time step before them. □

We want to show that it is impossible for the adversary to construct a valid ledger blockchain without investing enough time, even if she has access to sufficient secret keys. To show this, we will use the round-based PV game we presented in Section 3.1, with the difference that the prover has to provide both her PoW chain as well as her copy of the ledger chain. In this version of the game, the prover must add the random value x to the ledger chain. As we will see, this will require to rewrite the PoW chains as well. To confirm whether a pair of blockchains is valid $\mathcal{V}$ has access to all the verification functions presented in Section 4.1.2 as well as the knowledge of the current round. Due to the cheating recognition mechanism in the model, the adversary knows that any attempt at forking or rewriting the chain can result the participants under her control being eliminated from the protocol. This is true regardless of the amount of participants that the adversary controls. While we need a Q-majority of honest parties to ensure that the protocol works as expected, as long as there is one honest party, the adversary cannot create a fork without it being noticed. Note that this does not mean that the adversary cannot maintain a fork and keep it secret from the honest parties, only that whenever any honest party sees the fork, they will immediately recognise it and take action against the antagonistic participants.

For this model, the adversary $\mathcal{A}$ will take the role of the prover $\mathcal{P}$. For our game to be interesting, we have to change the setup, as rewriting the chain is impossible for the adversary. Especially when we consider that every block needs to be signed by enough participants. It is impossible for the adversary to construct a valid fork without fooling enough participants in order for them to sign a fake block. Even then, the participants will be eliminated from the protocol anyways for double signing blocks. The conditions of the protocol prevent the appearance of any type of fork. Therefore, we will have to change the setting in order for our PV game to be of any interest. We will then assume that the adversary has control of the whole network. The implication of this is that any security guarantee that we prove is independent from the public-key infrastructure. We will assume that the adversary gains control of the network at a certain point and then starts the process of rewriting the blockchain in some point in the past (as having control of the network, it can immediately make a polynomial number of forks starting from the block being created when $\mathcal{A}$ takes control). This setting is not purely theoretical, but also reflects a situation that can reasonably arise in applications. If a blockchain was subject to outside inspection, like in cases of government regulation, it might be in the participants' best interest to present a fake blockchain. Therefore, they could all cooperate to make a fake chain to present to the regulating agency. At the same time, they must continue to maintain the real blockchain so their operations continue as normal. In this section we will prove that even in this setting, it is impossible for $\mathcal{A}$ to rewrite the ledger chain if she does not have access to a faster rate. In the same way, we will show that if she does have this computational advantage, she will still need to invest sufficient time to be able to come up with a fake chain.

Lemma 4.15. *Suppose H and PoW are random oracles with security parameter λ and $\mathcal{A}$ and $\mathcal{V}$ are playing the round-based PV game for $\mathsf{IdealLipwig}^{\tau}$ with rate γ and n participants. For $\mathcal{A}$ to win, she must rewrite all n PoW chains starting from block $PC^j[i+1]$.*

Proof. In this setting, $\mathcal{A}$ has access to all private keys, so she can create valid blocks of all kinds. When adding x to $BC[i]$, $\mathcal{A}$ substitutes $BC^*[i]$ for $BC[i]$ such that $BC[i].link = BC^*[i].link$ and $BC[i].P = BC^*[i].P$. She follows this by substituting the pointers for $BC[i]$ to $BC^*[i]$ in both the ledger block $BC^*[i+1]$ and all PoW blocks from round $i+1$. Then, $\mathcal{A}$ changes the pointers in the proofs of work of $BC[i+1]$ to the new PoW blocks with the modified pointer and then changes the pointer of each block. Up to this point, the chains $BC^*[0, i+2)$ and $PC^{*j}[0, i+2)$ are valid without executing any proofs of work.

In the case that $r = 2$, $\mathcal{A}$ can just continue to run the protocol directly substituting the rewritten chains. Because they are of the right length, $\mathcal{V}$ will not notice the difference. Otherwise, the pointers in BC can be changed to the appropriate ones in BC^* so that $\mathsf{len}(BC^*) = i + r$. While we have that $\mathsf{verifyBC}\big(BC^*[0, k), BC^*[k]\big) = 1$ for all $k \in \{1, 2, \ldots, i + r\}$, we have that $\mathsf{verifyPC}(BC^*, PC^{j*}[k]) = 0$ for $k > i + 2$ and all $j \in \{1, 2, \ldots, n\}$. This is due to the fact that the proofs of work are computed over pointers to the original blocks. Therefore, in order for $\mathcal{V}$ to believe that the rewritten ledger chain is valid, all linked proof of work chains must be rewritten as well starting one block after the main chain was modified. □

The previous lemma represents how the immutability of the PoW chains is acquired by the ledger chain through having pointers that connect the chains. Note that it is not only the ledger chain which gains security, as all PoW chains also share the acquired immutability guarantees. In this case, the security of PoW does not change, as the strength of the proofs of work is the same. This will not be the case when we have participants with different rates. Thanks to the previous result, we can see that if the prover wants to win the PV game for $\mathsf{IdealLipwig}^{\tau}$, then she must win the PV game for $\mathsf{SingleLipwig}$ for all the PoW chains. We will use this fact to show the immutability guarantees of the ledger chain.

Lemma 4.16. *Given random oracles H and PoW with difficulty parameter λ. Suppose the ledger chain BC constructed in $\mathsf{IdealLipwig}^{\tau}$ has length $i + r$ (with $r > 0$), if the adversary $\mathcal{A}$ wants to rewrite $BC[i]$, it will take her no less than r rounds except with negligible probability in the security parameter λ.*

Proof. By the previous lemma, in order to rewrite the ledger chain in block i, $\mathcal{A}$ must rewrite all PoW chains at block $i+1$. To be able to do so, $\mathcal{A}$ must compute the proof-of-work function over the modified PoW blocks. This is equivalent to rewriting a the $i + 1$-st block of PoW chain in $\mathsf{SingleLipwig}^{\tau}$, which by Lemma 3.3 takes $r\tau\gamma$ time steps or r rounds.

□

If the adversary takes control of the network and tries to rewrite the chain without continuing the process of creating blocks, it will be clear that the protocol has been compromised. Therefore, the adversary must continue to run the protocol if she is interested in keeping the illusion that the protocol is working correctly. For example, if the blockchain represents a cryptocurrency the adversary is incentivised to hide the fact that she has control over the network because a compromised currency would lose its value. This implies that even when trying to rewrite the blockchain, the adversary must continue to grow the original chain. This will be counterproductive, as it will be computationally costly and, more importantly, will prevent her from ever accepting the modified chain.

Lemma 4.17. *In protocol $\mathsf{IdealLipwig}^{\tau}$, if an adversary has control of the network and wants to rewrite a block $BC[i]$, she must start computing proofs of* *work for the fork before round $i + 2$ or the modified chain will fail to reach the length of the original chain*

Proof. By Lemma 4.16, it takes an adversary $r - 1$ rounds to the r sets of proofs of work necessary to successfully rewrite the chain. If she starts at any round after $i + 1$, we will have $r \geq 2$, which means that it will take her at least one round to create the necessary proofs of work. For every block of the new chain BC^* that it creates, the original chain BC has grown by one block as well. Therefore, she will never be able to catch up with the original chain. □

The preceding lemmas show that the adversary cannot rewind the chain without halting the protocol. This implies that serial proofs of work make it effectively impossible to *rewind* the blockchain to a previous state to erase previous transactions. This would be of particular interest for chains that are subject to regulatory oversight. If the verifier receives a chain, it can be sure that the chain has been in existence for a certain amount of time. It is always possible that the network is maintaining two distinct chains since the beginning, but that implies serious computational cost, cooperation between all parties and prescience of things it would want to keep hidden in the future. Serial proof of work prevents the arbitrary creation of blocks even with control over the totality of the network, which is a guarantee that cannot be managed by protocols which do not use proof of work.

We now show that our ledger chain also fulfills $\theta(\gamma^*, r)$-security from Definition 3.9 thanks to the connections with the PoW chains.

Theorem 4.18. *Suppose H and* PoW *are random oracles with security parameter λ. A ledger chain BC created in* $\mathsf{IdealLipwig}^{\mathcal{T}}$ *is $\theta(\gamma^*, r)$-secure with*

$$\theta(\gamma^*, r) = \left\lceil \frac{r - 1\gamma}{\gamma^* - \gamma} \right\rceil - 1$$

Proof. In Lemma 4.15 we proved that rewriting a block in the ledger chain which is r blocks deep requires to rewrite all the linked PoW chains for $r - 1$ blocks. By Theorem 3.10 we know that this takes

$$\theta(\gamma^*, r) = \left\lceil \frac{r - 1\gamma}{\gamma^* - \gamma} \right\rceil - 1$$

rounds. Because all of the PoW chains can be computed in parallel, it takes at least that many time steps to rewrite BC. □

In this chapter, we have created a permissioned blockchain protocol that fulfills the properties necessary for its correct functioning. We have also shown how serial proofs of work provide this blockchain with immutability, by gaining the guarantees that appear in the PoW chains. However, many properties of this protocol are idealised and we will soften some of the assumptions in the following chapter.

The Lipwig^{ω} Protocol 5

We have built a protocol $\mathsf{IdealLipwig}^{\tau}$ in which multiple participants maintain a ledger chain in conjunction with their personal PoW chains. While we have shown this protocol to be secure, its structure is too rigid for any meaningful application. In particular, the supposition that rounds take a set time is too strong. Fixing the length of rounds requires strong assumptions over the rate of the participants as well as the worst-case scenario for the protocol. Furthermore, the round structure in $\mathsf{IdealLipwig}^{\tau}$ can only be maintained in a perfectly synchronous network. Otherwise, there is a possibility that blocks will not be created in time, which will make the protocol structure break down. By relaxing the length of rounds by allowing them to depend on the time needed in choosing a block, we present a setting that is closer to application. Not tying the execution of the protocol to a pre-set round length permits us to get rid of other unrealistic assumptions, in particular that all participants can compute proofs of work at the same rate. Without this supposition, we do not need to assume that all participants have the exact same equipment and the same conditions. We will base this new protocol on $\mathsf{IdealLipwig}^{\tau}$ but with ideas we developed in $\mathsf{SingleVarLipwig}^{\omega}$. The overall structure will resemble the former, but we will adopt the idea of a blockchain with proofs of work of variable strength from the latter. Having already worked with them, we know the immutability guarantees that we expect from the PoW chains remain in this new paradigm. However, other challenges will appear when we try to build our new protocol Lipwig^{ω}.

An issue when modelling rounds with a set length is the concern that the process of creating a new block might take longer than the current round. To prevent this in $\mathsf{IdealLipwig}^{\tau}$, we made sure that there would be enough time by making the rounds take considerably longer than BFT, as well as adding sufficient padding to make sure that everything terminates in time in the worst-case scenario. This solution has the side effect of lengthening the wait time for transactions to be added, as most transactions that happen in a certain round will not be written in the ledger until the next round. Therefore, whether the participants agree on a block quickly or not becomes slightly irrelevant as no advantage is gained by having an efficient algorithm for consensus. This seems like a waste, as the advantage of permissioned protocols over permissionless ones is that a stronger setting allows for more efficient consensus mechanisms. It would be desirable to take advantage of this fact and have each round finish whenever a block is chosen. On the other hand, if agreeing on a block takes a long time in a certain round, the protocol should not be affected.

An advantage of our serial proofs of work is that they provide a record of the length of the round encoded in the strength of our proofs of work. Then, if a round takes too much time to complete, we will have stronger proofs of work, as Golem ran for a longer time. This means that if a round lasts longer than expected, it will not affect the immutability of the blockchain. In Bitcoin, if a block takes a long time to find it does not imply that trying to rewrite the chain at that point will take any longer than at any other point. However, if throughout the process of choosing a block we have serial proofs of work being computed they will reflect the time spent computing it. Therefore, if we would want to slow down the creation of blocks, we could do so without it compromising immutability. This feature is something unique to our protocol and could be useful in a time of little activity in the network. For example, the volume of transactions might fall during the night[1] or during weekends. In a Nakamoto-based system, blocks must be created continuously in order to maintain the immutability guarantees, even if they are empty.

Our protocol has the flexibility of having rounds that can change length during the protocol. To have a way to signal when a block should be issued, we will define a functionality $\mathsf{Semaphore}$ which will communicate with Lipwig^{ω} to signal the end of each round. Whenever a block is completed, participants in Lipwig^{ω} will input it to $\mathsf{Semaphore}$, so it will know when a new round has started. In case we wish to model the system where rounds depend only on block creation, $\mathsf{Semaphore}$ will simply send a `halt` instruction at the beginning of every round. On the other hand, $\mathsf{Semaphore}$ can represent anything from a simple policy based on time of day to a complex system that signals the creation of blocks whenever certain external events happen. Untying the length of the rounds to the computation of serial proofs of work allows for participants to have proofs of work of varying strengths in their PoW chains. This fact will permit us to showcase another advantage of our construction that will form the building block of Chapter 6.

In the same way the ledger chain from $\mathsf{IdealLipwig}^{\tau}$ inherits the security from the PoW chains, each PoW block in Lipwig^{ω} will be strengthened by the strongest PoW block created that round. This fact will be especially relevant in the case PoW chains are used for a purpose beyond maintaining the ledger chain. While we do not focus on it in this presentation, PoW chains can be used as ledgers on their own. If a participant has an especially high rate, everyone

[1]Assuming the blockchain is deployed in an area with similar time zones.

else's PoW chains will benefit from it by inheriting the time-lock security from it. One might then ask what incentivises participants to compute the strongest possible serial proofs of work. We expect participants to want to secure the ledger chain as much as possible, something that is especially relevant in a permissioned setting, as all the actors are known. However, each participant may think that their contribution is not especially important and therefore not invest all the computational power they have available to compute proofs. However, as long as one participant uses his full power, the blockchain will be secure. We will see that this will be enough to act as an incentive for the rest of the participants to invest all their computational power on the computation of proofs of work, to avoid a tragedy of the commons. On the other hand, because we constructed our proof of work to be resource-efficient, only utilising one core, participants do not have a reason to avoid computing the proofs. Additionally, an incentive could be created for participants to contribute stronger proofs of work. For example, a participant could be randomly rewarded in a lottery where each party's winning probability is related to the strength of their proof. In this work, however, we will assume every honest participant will compute the strongest proofs of work they have access to.

This new protocol will not simply be an extension of $\mathsf{IdealLipwig}^{\tau}$ with looser constraints in the round time. We will present a provably secure permissioned blockchain protocol with assumptions that are reflected in reality. This new model will no longer require perfect synchrony in the network. We will also introduce the possibility of new participants joining the protocol at any moment.

After presenting this model, we will draft a extension of it in which we will enhance our blockchain with a randomness generator using our serial proofs of work. We do this not only to create an implementation of the public random beacon presented in the same paper from where we take our proof-of-work function [LW15] but also because of the importance of randomness for consensus. Many consensus protocols, like proof of stake, benefit from access to common randomness. Finding a way to create trustworthy randomness from inside the protocol without relying on any additional assumptions is another advantage of our construction. The security of the random-number generator only relies on assumptions that are already present in our model. While we will build Lipwig^{ω} from the same building blocks $\mathsf{Consensus}$ and BFT that we used in the previous chapter.

5.1 Constructing the Protocol

Our new protocol Lipwig^{ω} is built by modifying what we previously built for $\mathsf{IdealLipwig}^{\tau}$. We take advantage of composability by constructing a protocol that uses both $\mathsf{Consensus}$ and BFT as they were defined in the previous chapter.

Lipwig^{ω}

Initialization

- Receive (pk_j, sk_j),γ_j the list of pairs (k, pk_k) and ω from $\mathcal{Z}$
- Receive $BC[0], PC^j[0]$ and $\mathsf{Golem}(PC^j[0])$ from $\mathcal{Z}$
- Query for ledger chain, **if** non-empty wait until next ledger block is broadcast and start in following round

Round i

The round begins upon reception of $\mathsf{Golem}(PC^j[i-1], \gamma_j)$
PoW Chain Maintenance:

- Set $\sigma = \Sigma.\mathsf{sign}^j\big(H(PC^j[i-1]), H(BC[i-1]), \perp, \mathsf{Golem}(PC^j[i-1])\big)$
- Append $\big(H(PC^j[i-1]), H(BC[i-1]), \perp, \mathsf{Golem}(PC^j[i-1]), \sigma\big)$ to PC^j as $PC^j[i]$ and **broadcast** it
- Query for $\mathsf{Golem}(PC^j[i])$

On input `halt` from Semaphore

Pre-Consensus**, if** $j \in BC[i-1].\boldsymbol{NA}$**:**

- Let $comm = \mathsf{last}(BC).NA$
- After 2δ time steps: Let P be the set of every PoW block received with strength at least ω
- Call $P^* = \mathsf{BFT}^P(comm, \varnothing, P)$
- Input `receive` to TxsPool, get back Txs
- **Call** Consensus(BC, $comm \cap \mathsf{ind}(P^*)$, P^*, Txs)

On input `halt` from Semaphore

- **Input** `halt` to Consensus
- Wait until Consensus *outputs block*, *broadcast* it and add it to BC
- **If** a_j was not running Consensus, wait until a valid *block* is received and add it to BC
- Input `clean`($BC[i].T$) to TxsPool and input `done` to Semaphore
- **Input** `halt` to Golem and **receive** $\mathsf{Golem}(PC^j[i])$

Cheating Recognition

Whenever a_j receives a signed block B^* that is different from the one in the same stage B

- **Broadcast** (B, B^*)
- **If** Consensus is running and `halt` has not be input: input `cheat`(B, B^*) to Consensus
  Otherwise: Wait until the next instance of Consensus is running, then input `cheat`(B, B^*)

On **input** `chain`: Output BC
On **input** `PoWchain`: Output PC^j

As we can see now, many design choices for Consensus that might have seemed counter-intuitive or useless in the previous chapter are taken advantage of in this new protocol. For example, Consensus halts only after an instruction from the environment, something which could have easily been avoided in IdealLipwig$^{\tau}$, as the time when that command is issued is always the same. However, in this new setting, it is fundamental for the participants to have control over when Consensus halts and issues a block.

Beyond changes like this, it is clear that Lipwig$^{\omega}$ does not differ a lot from IdealLipwig$^{\tau}$. The differences are the following:

- The communication is no longer assumed to be synchronous, but now has a delay of at most δ time steps. The adversary is tasked with delivering messages and can order them in whichever way she wants as long as every message arrives within δ time steps from when it was sent. This fact makes us rely directly on BFT to determine the proofs of work that will be used in Consensus. Note that because we chose to construct BFT and Consensus in this setting in the previous chapter, there is no need to modify them.

- We have changed the initialisation conditions to not necessarily happen at the start. A new participant queries for the chain to see if the protocol has started and to see the current round. After this, they prepare to run in the next round.

- Similarly to SingleVarLipwig, we have added a parameter ω that sets a minimum strength for proofs of work. Because the participants have different rates, we cannot directly translate this bound into time steps. However, the fact that $\mathcal{Z}$ has access to the rates of the participants means that it is possible to ensure that rounds are long enough for proofs of work to be completed.

- We have added an independent ideal functionality Semaphore as a beacon to signal the end of a round.

- The parameter that determines a lower bound for the length of the round, ω, is a function of time and not a constant. Note that the use of ω is different than in SingleVarLipwig, where ω's role was simply to prevent rounds that are too small to prevent a particular attack.
- The committee that runs Consensus will be the participants who sent a valid proof of work. As we will see later, this fact will imply that the proofs will have to be of a certain strength relevant to the strongest proof that was issued that round.

There is a simplifying assumption that we make in this model that is of note. The G component in the PoW blocks is set to $\perp$, however, any participant may add whatever he wants to it, in the style of SingleVarLipwig. We let it be $\perp$ for simplicity, as otherwise we would have to define where the information comes from. If we permit participants to add anything to G instead of $\perp$ does not change the security of the model.

5.1.1 Variable Strength

In Section 3.1.2 we had presented a protocol where rounds could last as much time as the participant wanted. This setting implied that the proofs of work would be stronger as more time passed. This construction did not require us to change many things of SingleLipwig. Unfortunately, doing this with more than one participant implies issues which do not exist if there is only one participant. Having multiple participants with different rates requires us to add parameters to ensure the correct working of the protocol. These new issues stem from the fact that we cannot use strict parameters, as we must account for different rates as well as the delay in communication.

The first thing that will change is the definition of the minimum strength parameter ω, as we must take into account the fact that not all participants can compute a proof of the needed strength in the same time. To take that into account we will define two parameters, γ_{min} and γ_{max}, that will bound the rates of the participants in the protocol. Given the lower bound for the rates we will define the minimum strength of the proofs of work as follows:

Definition 5.1. *In the protocol* Lipwig$^\omega$ *given γ_{min}, the* ***minimum strength parameter*** *ω is a function over time steps t such that*

$$\omega(t) = \begin{cases} \omega^* & \text{for } t < t^* \\ t\gamma_{min} & \text{otherwise} \end{cases}$$

where $w^* = t^*\gamma_{min}$ and $t^* \gg 2\delta$.

This function says that there is a minimum strength for all proofs of work ω^*. Once enough time passes for everyone to create a proof of strength ω^*, the minimum strength grows at a constant rate based on γ_{min}. The first thing to take into account with this definition is the fact that there is no direct way to encode the length of a round. Without knowing the rate of a participant, we cannot know the length of each round. Even then, if we cannot ensure that the participant computed the proof of work during the whole round, the proof of work gives us only a lower bound on the length of the round. Even from the perspective of each participant, the length of the round for other participants is not perfectly known. Due to the delay of δ time steps on messages, the length of the same round can differ by 2δ time steps according to different participants. This delay seems to create an issue with actually enforcing that proofs of work are correct when one sees the chain, as one needs to know t to determine the value of ω. Here is where the bounds over the rate can help us. Given a ledger block $BC[i]$, let j be the index of the strongest proof of work found in $BC[i].P$. We let the argument of ω be $\mathsf{str}(PC^j[i])/\gamma_{min} + 2\delta$. While this is not the *real* minimum, it is close enough for our purposes. More importantly, it does not require to know the actual rates of the participants, but only the bounds. We will enhance BFT^P and our verification functions with the ability to recognise proofs of work that have the requisite length. In particular, $\mathsf{verifyBC}$ must check whether the proofs of work are of the requisite strength. Because this depends only on the blockchain, the way to check if the proofs of work are of the expected strength is by taking the strongest one, dividing it by the maximum rate and subtracting 2δ to get t'. Every proof of work in the block would then be of strength $\omega(t')$.

When an honest party is initialised in Lipwig^ω, it is given its rate $\gamma_j \in [\gamma_{min}, \gamma_{max}]$. This variable represents the power of the core that the participant will assign to computing the proof of work. The rate determines how fast proofs of work can be computed by each participant. Not having a fixed rate introduces a potential complication: what happens if some participants are not able to compute valid proofs of work within one round? If a participant's rate is too slow or the rounds are too short, some participants may not be able to contribute a proof of work to the chain. We will demand a minimum strength for proofs to be considered valid and added to the chain, which might further complicate this problem. We will ensure that any compliant execution of Lipwig^ω allows for rounds that are long enough for participants to compute a proof of work that is long enough.

In the permissioned setting it is reasonable to assume that we know the rates of the participants. If this were not the case, any participant could get a lower bound on any other participant's rate after they have contributed proofs of work. The adversary's rate, $\gamma_{\mathcal{A}}$, is as fast as the fastest corrupted participant's rate. That is, all adversarial participants can call $\mathsf{Golem}_{\mathcal{A}}$ even if originally (before being corrupted), their rate was lower.

For the protocol to run correctly it will be necessary for the rounds to be long enough that sufficient participants can create proofs of work of the requisite strength. Enforcing this restriction means that a round must last more than t^* time steps, with t^* being the value in the definition of ω. We will force every round to be at least as long as it is needed for enough honest parties to finish computing their proofs of work before the round ends. We are comfortable enforcing this because in practice a minimum strength should be chosen in function of the expected time for consensus. The purpose of ω is just to ensure that participants are actually computing the proof of work as expected. While the minimum strength is enforced by the participants, in our extension of the model presented in Section 5.3, we will show a way to enforce it in a more direct manner.

We will use Semaphore to determine the length of the rounds in Lipwig^{ω}. As mentioned earlier, Semaphore is purely an abstraction of the conditions that we use to determine the length of rounds. Therefore, we will treat it as an ideal functionality that is only concerned with sending orders to stop the round. We will treat Semaphore as a central, public, independent and authenticated beacon that can communicate with each participant instantly, that is, the network delay δ does not apply to messages to and from Semaphore. At the same time, it is impossible to access the history of the messages sent by Semaphore to try to find the history of the creation of the blockchains. Therefore, while we have very strong assumptions over Semaphore, the security of the protocol does not depend on it.

One important difference between $\mathsf{IdealLipwig}^{\mathcal{T}}$ and Lipwig^{ω} is that the latter allows for participants to join at any time. When a participant joins the protocol, he must first query for the current chain. He must have some assurance that the chain that he received is the correct one and not generated by the adversary to confuse him. This is a problematic event in certain protocols, in particular protocols which do not use proofs of work to directly create consensus like [PS16b, DPS16]. Systems based on proof on work do not have these issues, as it is computationally unfeasible for the adversary to maintain valid forks [Nak08, GKL14]. Our protocol's use of proofs of work makes maintaining forks complicated for the adversary, the fact that the blocks are signed makes it almost impossible. Even if the adversary were to take control of the network, it cannot arbitrarily create a fake blockchain that a new participant would accept[2]. Therefore, a new participant will be able to identify the correct chain unless the adversary has control of the network for a long enough time. In that case, this is effectively irrelevant as the whole protocol has been compromised.

5.1.2 Security Proof

We must now prove that the protocol is secure.

Given the protocol, we must characterise a **compliant execution of** Lipwig^{ω}. Given an environment-adversary pair $(\mathcal{Z}, \mathcal{A})$ and a minimum strength function

[2]While we have not proved this explicitly for this protocol yet, this can be seen as a combination of Theorem 3.11 and Lemma 4.16

ω with parameters ω^* and t^*, we say the pair is $(n, Q, L_{\mathsf{BFT}}, \delta, \gamma_{min}, \gamma_{max})$**-valid** w.r.t. Lipwig^ω if the following properties hold:

Bounded Network Delay Every message output by an honest participant reaches every other honest participant after at most δ time steps after it was sent.

Timely Initialisation Before the start of the protocol, parties are initialised by $\mathcal{Z}$. Before they are initialised, the adversary might communicate to $\mathcal{Z}$ which parties it wishes to corrupt. These parties will start the protocol already corrupted, as long as they constitute no more than $(1 - Q)n$ participants. At the beginning of any round, new participants may be created by the environment.

Valid Initialisation At the beginning of the protocol, every participant receives the same correctly constructed $BC[0]$ and also receives valid $PC^j[0]$, $\mathsf{PoW}(PC^j[0])$, $\gamma_j \in [\gamma_{min}, \gamma_{max}]$ and a pair of keys (pk_j, sk_j) that are shared by no other participant. They also receive the correct list of association between indices and public keys.

Adaptive Corruption At any round $\mathcal{A}$ might send a message $\texttt{corrupt}(a_j)$ to $\mathcal{Z}$. At the beginning of the next round, $\mathcal{A}$ will gain control of a_j as long as that means that no more than $(1-Q)$ of the active participants in that round are being controlled by $\mathcal{A}$.

Timeliness Every participant queries Golem within δ time steps

Minimum Run Every round must last enough time for all honest participants to contribute a block. If there is at least one participant with rate γ_{min} this is equivalent to saying that all rounds must last at least t^* time steps.

Definition 5.2. *Given random oracles parametrised by λ, Lipwig^ω is said to be secure against $(1-Q)$-corruption with liveness parameter L_{BFT} if for any $n > 0$ and for any pair $(\mathcal{Z}, \mathcal{A})$ that is $(n, Q, L_{\mathsf{BFT}}, Q, \gamma_{min}, \gamma_{max})$-**valid** w.r.t. Lipwig^ω, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, except with $\mathsf{negl}(\lambda)$ probability, the following properties hold for $\text{EXEC}_{(\mathcal{Z},\mathcal{A})}[\mathsf{Lipwig}^\omega]$:*

Consistency *: If an honest participant a_j is queried $\texttt{chain}$ at time t and outputs BC and an honest participant a_k (possibly equal to a_j) is queried $\texttt{chain}$ at time $t^* \geq t$ and outputs BC^*, we have that $BC \prec BC^*$ or $BC^* \prec BC$. If $t^* \geq t + \delta$ or $a_j = a_k$ we have that $BC \prec BC^*$*

Liveness *: If a valid transaction tx appears in $\mathsf{TxsPool}$ in round i then either $tx \in BC[i]$ or $tx \in BC[i + 1]$*

Lemma 5.3. *Given a compliant execution of Lipwig^ω, every execution of BFT by Lipwig^ω is compliant.*

Proof. We need to fulfill the following properties:

Bounded Network Delay By *Bounded Network Delay* of Lipwig^ω

Initialisation Agreement BFT is initialised with $comm = BC[i-1].NA$ and $hist = \varnothing$. In Round 1 there is agreement in the genesis block by construction. In any other round, if Lipwig^ω is secure then everyone agrees on $BC[i-1]$ and therefore on $BC[i-1].NA$.

Static Corruption Follows from *Adaptive Corruption* of Lipwig^ω, because participants can only be corrupted at the beginning of each round and at least Q of the active participants are honest.

Minimum Run By construction, Lipwig^ω can only continue after BFT^P finishes running.

Close Start and Stop Whenever a block is created, it is broadcast. As it is created with the requisite signatures, getting a block from only one person is enough to know that it is valid. Let a_1 be the earliest honest participant to get block $BC[i-1]$, lets call this time 0. He immediately broadcasts it and starts the next round. It takes two time steps for him to reach the time when he needs to wait for other participants, he waits for 2δ time steps, until time $2\delta + 2$, where he queries BFT^P. Meanwhile, the latest time an honest participant will accept a new block is time δ. Then they will start the next round in the same way as a_1, which means they will call BFT^P in time $3\delta + 2$, within the limits. Honest parties will never halt BFT^T, so *Close Stop* is trivially true.

$\square$

Lemma 5.4. *Given a compliant execution of* Lipwig^ω*, every execution of* $\mathsf{Consensus}$ *by* Lipwig^ω *is compliant*

Proof. We need to fulfill the following properties:

Bounded Network Delay Follows from Lipwig^ω's **Ideal Network Properties**

Initialisation Agreement $\mathsf{Consensus}$ is initialised with the following variables that must match:

- **BC**: Every participant has the same genesis block from *Valid Initialisation* of Lipwig^ω. So in Round 1, this holds. Every other block is generated by $\mathsf{Consensus}$. Because of $\mathsf{Consensus}$'s *Agreement*, by induction, every honest participant always holds the same BC.
- *comm*: As seen in the previous point, every honest participant holds the same BC, so $\mathsf{last}(BC).NA$ is always the same. On the other hand, P^* is agreed upon using BFT, hence by *Agreement* of BFT, everyone holds the same P^* (We can count on BFT fulfilling Agreement due to Lemma 5.3). Therefore $\mathsf{last}(BC).NA \cap P^*$ will be the same for all participants.

- **W**: Directly by BFT's *Agreement*

Valid Initialisation The following inputs must fulfill certain properties

- **BC**: The first ledger block is correct by construction, assuming that Lipwig$^{\omega}$ is secure, every following block must be valid by induction
- *comm*: As seen in the previous point, every honest participant holds the same BC, so $\mathsf{last}(BC).NA$ is always the same. On the other hand, P^* is agreed upon using BFT, by *Agreement* of BFT, everyone holds the same P^*. Therefore $\mathsf{last}(BC).NA \cap P^*$ will be the same for all participants.
- **W**: BFT^{P} uses verifyPC to ensure that every PoW block is valid. *Minimum Run* ensures that everyone can contribute a proof of work. However, we have to prevent the possibility that the adversary starts computing the proof of work earlier than she should in order to have the proofs of honest parties counted as invalid as they are not strong enough compared to theirs. However, because the round ends when a block is completed (and is guaranteed to end within δ time steps for every honest party), the adversary can only gain an advantage of δ time steps to compute proofs of work. However, verifyBC allows a 2δ cushion for proofs of work, which negates this advantage.

Static Corruption Lipwig$^{\omega}$'s *Adaptive Corruption* ensures that no participants become corrupted during the execution of Consensus.

Close Start and Stop Close start comes from the fact that after the execution of BFT^{T} there is no longer any interaction needed before calling Consensus. By *Close Start and Stop* of BFT, every party finished BFT within δ time steps of each other, therefore they will start Consensus within the same time. Close stop follows from the fact that Semaphore sends the `halt` output to everyone at the same time.

Cheating Timeliness By construction, if cheating is discovered after a `halt` command in a round, honest participants wait one round before inputting `cheat`.

□

Theorem 5.5 (Lipwig$^{\omega}$ from Consensus and BFT)**.** *Suppose that the signature scheme is unforgeable, that H and* Golem *are independent random oracles parametrized by λ. Let N_i be the set of participants at round i. Suppose that* Consensus *and* BFT *are secure against $(1-Q)$-corruption with liveness parameter L_{BFT} for $Q > 1/2$. Then* Lipwig$^{\omega}$ *is secure against $(1-Q)$-corruption.*

Proof. **Consistency** : Whenever any participant adds a block to the chain, it comes from Consensus. By the *Agreement* property of Consensus, this will also be the same. By construction of Lipwig$^{\omega}$, every participant will get a new block within δ time steps.

Liveness : Suppose a valid transaction tx appears in TxsPool in round i, there are two cases:

- tx **appears before querying** TxsPool: tx is continued in the Txs pool that is input into Consensus. *Liveness* in Consensus implies that tx will be contained in the block issued at the end of the execution of the subprotocol.
- tx **appears after querying** TxsPool: tx remains in TxsPool and is not eliminated when the participant inputs the accepted block. Therefore, in the next round, when TxsPool is queried, tx will be in the set it outputs. Then we follow the previous proof to show that $tx \in BC[i+1]$

□

We have proved that this new protocol is secure, taking care of the new details introduced by the rounds of variable length and the minimum strength of the proofs of work.

5.2 Immutability

In previous chapters, we define immutability through the eyes of a verifier who is an outsider. The idea behind this was to show that the structure of the protocol itself was good enough to prove immutability, even to someone who is not participating in the protocol. An assumption that we did in the previous chapters was that the verifier knows the rates of the participants. This permits him to directly confirm that the proofs of work were computed correctly. However, in this new setting we are not able to do that, as we allow players to have varying rates. Additionally, delays in communication can make the interval between each block different for each participant. Therefore, we must be content by having looser bounds for the guarantees in Lipwig$^\omega$. On the other hand, the setting will alow us to strengthen the security of the PoW chains of participants with a slower rate.

The structure of Lipwig$^\omega$ is very similar to IdealLipwig$^\tau$ and therefore similar immutability guarantees will hold for the ledger chain in Lipwig$^\omega$. However, there are many new details to take care of. We have already seen similar PoW chains in Section 3.1.2 and our proof will mirror them the same way the proofs for IdealLipwig$^\tau$ were based on the ones in Section 3.1.1. First, we will show that these blockchains are time-locked not only by their own proofs of work, but by the strongest proof created in each round. We will have to be careful defining our parameters to be able to prove immutability with the additional concerns.

In this new setting we will see that the guarantees of PoW chains are strengthened by being part of Lipwig$^\omega$ (as opposed to being seen as individual instances of SingleVarLipwig$^\omega$). In Lipwig$^\omega$, not only do the PoW chains secure the ledger chain but they also strengthen each other's security through it. The protocol is built in such a way that if a participant wants to have any say in the structure of the ledger chain, he must submit a valid proof of work in a signed PoW block. This fact gives every participant a record of everyone else's PoW chain in the form of a record in the blockchain. The cheating prevention mechanism will then prevent anyone from changing their PoW chain. Acting as a ledger for commitments of all PoW blocks is not the most important role of the ledger chain. Due to the way we construct it in Lipwig^{ω}, the ledger chain makes every block (including the blocks in the ledger chain itself) as strong as the strongest block of the round. This relation implies that a block is time-locked not by its own proof of work, but by the strongest combination of proofs by all participants. To do this, we will slightly abuse notation by letting $\max_j\{\mathsf{str}(BC[i].P)\}$ represent the strength of the strongest proof of work encoded in block $BC[i]$.

Theorem 5.6. *Suppose we are running* Lipwig^{ω} *with H and* PoW *parametrised by λ and our ledger chain BC and all the PoW chains PC^j are of length $i+r$, with $r > 0$. Any PoW block $PC^j[i]$ is time-locked for γ_{max} by*

$$\sum_{k=1}^{r}\left(\frac{\max_j\{\mathsf{str}(BC[i+k+1].P)\}}{\gamma^*} - \delta\right)$$

where γ^* is the fastest rate that a participant could have access to.

Proof. We will prove this by induction on the distance from the last block.

All PoW blocks $PC^j[i+r-1]$ from round $i+r-1$ had to be created at least ${}^{\mathsf{str}(PC^j[i+r])}/_{\gamma_{max}}$ time steps earlier, which are the steps necessary to compute the PoWs in $PC^j[i+r]$. Because these blocks are represented in $BC[i+r-1]$, these blocks were created within δ time steps of each other. Therefore, they were all created at least ${}^{\max_j\{\mathsf{str}(BC[i+r].P)}/_{\gamma_{max}} - \delta$ time steps ago.

For the induction step, we know that $PC^j[i-1]$ was created before $PC^j[i]$ by construction. By our induction hypothesis, $PC^j[i]$ must have been constructed at least $\sum_{k=1}^{r}\frac{\max_j\{\mathsf{str}(BC[(i+1)+k+1].P)\}}{\gamma_{max}} - \delta$ time steps ago. Additionally, we can see that $PC^j[i]$ took at least ${}^{\max_j\{\mathsf{str}(BC[i].P)\}}/_{\gamma_{max}} - \delta$ time steps to be created. Therefore, $PC^j[i]$ is time locked by $\sum_{k=1}^{r}\frac{\max_j\{\mathsf{str}(BC[i+k+1].P)\}}{\gamma_{max}} - \delta$.

□

Note that this proof represents only a lower bound on how long ago the block was created. For certain blocks, this bound can be tightened, in particular for the block with the strongest proof of work of each round, as in that case we can get rid of the δ parameter. We have shown that PoW chains can enhance each others' security through their connection in the ledger chain. Now, we will show that the ledger chain is time locked by the same mechanism.

Theorem 5.7. *Let BC be a ledger chain created in* Lipwig^{ω} *with H and* PoW *parametrised by λ such that $\mathsf{len}(PC) = i+r$ with $r > 1$. Any ledger block $BC[i]$*

is time-locked for γ_{max} by

$$\sum_{k=1}^{r-1} \frac{\max_j\{\mathsf{str}(BC[i+k+2].P)\}}{\gamma_{max}} - \delta$$

except with negligible probability in the security parameter λ.

Proof. A block $BC[i]$ is pointed at by every PoW block $PC^j[i+1]$. Therefore, $BC[i]$ must have been created before these blocks were. By Theorem 5.6, each one of these blocks is time locked by

$$\sum_{k=1}^{r-1} \frac{\max_j\{\mathsf{str}(BC[(i+1)+k+1].P)\}}{\gamma_{max}} - \delta$$

Therefore, so is $BC[i]$. □

Similar to what we did in the previous chapter, we will now show that the PV game from Section 3.1.2 for Lipwig^{ω} has multiple PV subgames for $\mathsf{SingleVarLipwig}^{\omega}$. Before we do this, we must discuss the differences between these two protocols. As mentioned previously, the verifier does not know the rate of any participant, so he must estimate it. This gives the adversary an advantage, as she can compute proofs of work that are weaker than they should, and the verifier has no way of knowing. Additionally, delay in communication means that rounds may have different lengths for different participants, which also provides the prover an advantage. On the other hand it is important to note that while the communication in the protocol may be delayed, the communication in the game is immediate. As we are assuming that all participants are colluding, we will assume that there is no delay in the communication and the rounds start and end for everyone at the same time. The clocks of the prover and the adversary are synchronised so there is no ambiguity in the times t_0, t_1 and t_2 that form the base of the PV game. We also assume that the prover has the power to end a round at any time (that is, this does not depend on Semaphore) so she is able to start computing a new block the moment she receives x from $\mathcal{V}$. This gives the prover slightly more power than she would have in the regular execution of the protocol, but these modifications help to simplify the work without noticeably affecting the result.

We will make an additional assumption to simplify the notation, that when the prover receives x, she adds it to $BC[i]$ and immediately starts computing the proof of work over $PC^j[i+1]$ which she stops at time $t_2 - 1$. While she can add multiple blocks instead of only these two, it will make no difference in the time necessary. As a matter of fact, it is slightly more advantageous to follow this strategy, as in any other strategy she could lose strength by repeatedly stopping Golem at the middle of a computation.

Definition 5.8. *Given a PV game for* Lipwig^{ω} *with n total participants, a verifier $\mathcal{V}$ considers a blockchain set $\{BC\} \cup \{PC^j | j \in N\}$ to be* ***valid*** *if* *the random value x is found in $BC[i]$ then, for all PoW chains in the set:* $\mathsf{str}(PC^j[i+2])/\gamma_{min} + 2 \geq t_2 - t_0$ *in addition to enough chains fulfilling* verifyPC *and* verifyBC*.*

Unfortunately, as we do not know the rate of the participants, we must assume that they have the weakest acceptable rate. The verifier can be more demanding if he happens to know the rate of any participant. If this is the case, he can ensure that the PoW chain of that participant was computed according to that rate instead of γ_{min}. Additionally, he can use this rate to estimate the rates of the rest of the participants.

It is clear that the results of Lemma 4.15 also apply in this setting: a change in the ledger chain implies a change in every PoW chain starting from the next block. Therefore, the PV game for Lipwig^{ω} can only be won by the prover if she also wins all the $\mathsf{SingleVarLipwig}^{\omega}$-PV games for each PoW chain, assuming that all participants have rate γ_{min}. In these subgames, it is not necessary to take into account

Theorem 5.9. *Let H and* PoW *be random oracles with security parameter λ. The ledger chain created by* Lipwig^{ω} *is $\theta(\gamma^*, t_1)$-secure where*

$$\theta(\gamma^*, t_1) = \frac{\gamma^* t_1 - \gamma_{min}(t_0 + 1)}{\gamma^* - \gamma_{min}} - 1$$

Proof. Follows directly from Theorem 3.14 and the transformation presented above. □

By knowing the rate of a participant, the verifier can strengthen Theorem 5.9 by using this rate instead of γ_{min} and therefore have a stronger guarantee. Additionally, if the verifier has access to blocks that were computed before the challenge, knowing the rate of one participant will let him estimate the rate for all the rest, in order to build a harder PV game.

Lemma 5.10. *Suppose that H and* PoW *are random oracles with security parameter λ. Suppose a verifier $\mathcal{V}$ knows the rate γ^* of a participant a^* who is running* Lipwig^{ω}*. Then, in round i, the verifier $\mathcal{V}$ knows that for every participant a_j*

$$\gamma_j \geq \gamma_j^i = \max_{0<k\leq i} \left\{ \frac{\mathsf{str}(PC^j[k])}{\mathsf{str}(PC^*[k])/\gamma^* + 2\delta} \right\}$$

Proof. Choose r such that

$$r = \arg\max_{0<k\leq i} \left\{ \frac{\mathsf{str}(PC^j[k])}{\mathsf{str}(PC^*[k])/\gamma^* + 2\delta} \right\}$$

According to a^*, round r lasted $\mathsf{str}(PC^*[k])/\gamma_{\mathcal{V}}$ time steps. By construction of Lipwig^{ω}, the longest it could have lasted for a_j is 2δ time steps more. In that case, the minimum rate that a_j had to have to compute the proof of work in $PC^j[r]$ is at least the value of the quotient over which we maximise r. □

We have shown that the security guarantees that we expect from the blockchain still hold in Lipwig^{ω}. More importantly, we have seen how the structure of the protocol allows PoW chains to become more secure thanks to their connection to the network. Chains maintained with participants with lower rates benefit from participants having a faster rate. The same mechanism that secures the ledger chain secures each individual PoW chain. We have shown that Lipwig^{ω} creates blockchains that cannot be arbitrarily modified, even through collusion by all participants. We have also shown that anything encoded in any of our blockchains is mathematically guaranteed to have been created a certain amount of time ago. This guarantee is true even when there are no trusted parties, meaning that we extend the results in [BHS93]. By using the blockchain as a timestamping mechanism, we have gone full circle and used the basic hash-chain structure for the goal for which it was originally proposed.

5.3 Random Number Generation

We have built a secure blockchain using serial proofs of work, but we can do more with it. We chose to use a function which was previously used to generate public, verifiable randomness to act as our serial proof-of-work function. Can we also use it for its original purpose taking advantage of the blockchain? The idea behind the *unicorn* construction in [LW15] is to create a source of randomness that anyone can take part in but no one can control. This vision sounds very similar to the idea behind blockchains, in which different participants collectively create a ledger that they can all trust. Considering we are using similar tools and assumptions, could it be possible to combine both of these ideas to realise both systems? The answer is yes.

Our protocol Golem is an abstraction of the *sloth* function presented in [LW15], which is the fundamental ingredient in the generation of random numbers. The basic idea is to seed that function with information that can be contributed publicly. The protocol presented in that paper contemplates a website where information can be contributed by anyone. After some time, this information is combined with some additional non-controllable information (which is also public) to add entropy and then *sloth* is run for a previously defined number of iterations. This function allows to create a random string through the guarantee that the only way to predict the output is by computing *sloth* faster. Our ledger blocks fulfill all the properties which we desire from the seed. The components of the block are contributed by everyone and both proofs of work and signatures cannot be controlled by the participants. Proofs of work are the output of a random oracle and signatures depend on the block that is being signed as well as a previously assigned key. There is no need to add extra information to create entropy, as the only thing that can be controlled directly by the participants are the transactions. Even if the participants wanted to influence the random number by the choice of transactions, they can only do this without knowing the signatures that will be added later. Therefore, they cannot introduce information that they have full control over. Even if they could add some information over which they have complete control, it has been shown that this is not enough to influence the randomness of the output.

Simply running Golem is not enough to create randomness over which all participants will agree. The number of iterations of PoW must be publicly known and agreed upon by everyone. It should also be large enough to prevent the possibility of precomputation. A good candidate for this value is ω^*, the parameter found in our minimum strength function ω. The value for ω^* is encoded into the protocol and cannot be modified by anyone[3]. Using the ledger block as a seed evades most of the issues in the original implementation of *unicorn*. In the setting of the original paper, it is important to carefully set the intervals in which information can be contributed, something that we have automatically in a round. More importantly, because a block is only signed at the end of the round, no one has access to the completed block earlier than they should. At the same time, because the last things added to the blocks are signatures, it is impossible to predict them, as this will be effectively a forgery. This means that any attempt to control the contributions to the block in order to influence it will be incomplete. With this, we have a setting that mirrors the one presented in [LW15]. Therefore, we can use those results to show that generating random numbers over Lipwig^{ω} is secure.

We have shown that Lipwig^{ω} can create a source of randomness that realises the expected properties of *unicorn*. Our interest in creating this randomness is not based solely on the similarities between our proof-of-work function and *sloth*, but also because randomness can be especially relevant in the consensus stage of a blockchain. Implementations of byzantine fault tolerance that are robust in asynchronous networks rely on a leader who determines a candidate block over which the other participants agree [CL+99]. The choice of the leader can be done in many ways, but using randomness over which all participants can agree upon is a particularly simple and efficient way. Byzantine fault tolerance is not the only protocol that benefits from randomness. Agreeing on a random string is an essential part of proof-of-stake consensus, where block issuers are chosen by a lottery where participants have different probabilities depending on the amount of money they control [BGM16]. Implementations generate this randomness in different ways [DPS16, KRDO17, Mic16] with different levels of complexity and security. The advantage of our system is that no additional assumptions are required to trust the randomness, as it is based on the same building blocks of the protocol.

Using the randomness created from each block in the creation of the next one has the additional advantage of indirectly enforcing the *Minimum Run* condition of Lipwig^{ω}, as it would be necessary to compute the random string first

[3]In the case of the protocol in Chapter 4 which does not have an ω, we could use $\gamma\tau$ instead of ω

to be able to create a new block. Unless the gap between γ_{min} and γ_{max} is too high, this will probably ensure that the round lasts long enough for everyone to create a valid proof of work.

One might ask whether this random string is in fact a proof of work over the ledger blocks. This question is especially relevant if the string is going to be encoded in the block. If we have a proof of work for the ledger chain in the form of the random string, is it really necessary to have PoW chains to secure the ledger chain? The problem with relying on this random string to offer security is that it will not encode the total time needed to create each block, only enough time to iterate PoW ω^* times. This means that there will be potential to waste time. On the other hand, trying to generate randomness without a fixed number of iterations in mind introduces a consensus problem and, more importantly, a way for a dishonest participant to manipulate the randomness (by stopping at an iteration where the string has a desired property). Therefore, while the random string does add a level of security to the ledger chain, it is incomparable to the security provided by the PoW chains.

5.4 Practical Considerations

The current model contains multiple idealising assumptions that might not reflect the practicality of the actual implementation. While new use cases for blockchains appear every day, there are many challenges that must still be overcome. In particular, scalability continues to be a primary concern [CDE$^+$16]. Due to the permissioned nature of our model, we do not fall into most of the pitfalls related to blockchain scalability. However, recent advances in permissioned blockchains, particularly Hyperledger's Fabric [Hyp17], have introduced some design choices for a more practical application. Our design decisions are made to present a simplified model to prove security, but can (and should) be modified in an implementation. Here is where the choice of universal composability shines, as having a modular protocol permits us to change the subprotocols used in the protocol without having to prove security for the new system. In this section we will briefly explain how to make slight modifications to our model in order to accommodate the scalability concerns and suggest a more practical implementation of our model.

The model that we have presented attempts to describe the abstraction behind a blockchain protocol without focusing on the practical realities. Therefore, we make several simplifying assumptions which need to be translated into a practical setting, which might imply some complications. For starters, we know that our assumption that PoW's domain is the range of H does not translate into reality, as we want the domain of PoW to be considerably larger[4] than the range of our hash function. Solving this is simple, by splitting the information

[4]Eight times larger, to be exact.

to be hashed in eight distinct parts, hashing and concatenating them. However, a more interesting solution would be to use Merkle trees to hash the block, with the root of the tree taking the place of H where the partition of the block in eight creates the leaves. This would allow for a way to have hash pointers that can be reconstructed from proofs of work. Another change of the model will be regarding the use of signatures. We assume a straightforward public key infrastructure, where every participant signs each block, making the number of signatures grow with the network. There are different ways to solve this issue, either by limiting the participants who need to sign a block or by using more more theoretically interesting constructions, like group signatures [STV+16].

While the protocol we presented uses the same protocol to agree on signatures, proofs of work and transactions for simplicity, this is something we do not need, and probably do not want, to follow in an implementation. Agreeing in transactions requires creating consensus over which transactions should be accepted and which should not, which might be more complicated than making sure everyone received all of the signatures or proofs of work. Therefore, we can use simpler (and faster protocols) for those instances and use a more robust protocol to agree on transactions. An important detail that we did not model in our protocol is the validation of transactions, assuming all the transactions that players get are valid. This is a departure from the Bitcoin model, where the block creators must determine the validity of each transaction before adding it to a block. However, this idea is in line with newer proposals where transaction validation and block creation are done by different participants. Effectively, our protocol has a similar system in mind which we treat as a black box. In the same way that we can plug in different consensus protocols, we can add different validation methods in place of TxsPool, depending on what we want from it.

Having different participants executing different roles in the protocol is one solution for the scalability problem. Due to the trust assumptions in permissioned networks, it is more natural to have only a certain subset of participants performing certain roles (although there certainly are permissionless models that also do this, like [KKJG+17]). In particular, more recent implementations of permissioned blockchains have only a subset of participants selecting and ordering transactions. With the building blocks we have presented in this thesis, it is possible to create a variant of Lipwig^{ω} where only a certain subset of the participants participates in the creation of new blocks. We only need to have a way to select our subset, adapt our adversarial model accordingly and add a couple of superficial changes to Consensus and verifyBC to be able to create a protocol with the same security guarantees that Lipwig^{ω} provides and with higher potential for scalability.

Another problem with scalability in blockchains is the fact that the ledger can become too long; in the time of writing of [CDE+16] it took four days to download the entire Bitcoin blockchain. Saving the whole chain requires storing a large amount of information which may be unnecessary, but is necessary to ensure the integrity of the blockchain. While some proposals exist to prevent this constant growth of the blockchain, like in [Pev17], implementations seem to be going in the direction of participants only maintaining the parts of the ledger that are relevant to them. Taking into account the version of Lipwig$^{\omega}$ where only some participants add new blocks to the ledger chain, those same participants could be the only ones who maintain the ledger chain in its entirety while the relevant information is stored in each participant's PoW chain. This way, every participant is still actively contributing to the protocol while maintaining only their chain. Any issue that might require checking a ledger block can be achieved by querying for a specific block to the maintainers. Then the participant can confirm the correctness of this block by the hash pointer found in the relevant PoW block. This system could also help enforce privacy, with the transaction information in the ledger encrypted and only accessible to the parties who participate in each transaction, who would then maintain their own ledger in their PoW chains.

The use of PoW chains can extend beyond just being a view of the current state of the blockchain. They can be used to *pre-commit* transactions in such a way that they must be added to the next ledger block. They can even act as connectors between two different ledger chains, acting as a bridge sharing security and possibly even transactions. In the next chapter we will see how personal proof of work blockchains can do much more than secure a traditional blockchain. PoW chains can be the building blocks that help create a web of trust between peers in the internet.

Web of Trust 6

The thesis up to this point has focused on a blockchain as a distributed ledger over which many participants achieve consensus. The appearance of the blockchain in Bitcoin has associated the idea of a blockchain with a transaction ledger to support a cryptocurrency (or more recently, as a platform for smart contracts). However, this is not the only arena where a blockchain can be useful. In the business world there has been growing buzz over the possibilities of blockchain in areas so different as supply chain and land registries. Many of these applications are seen as ill fitting, as they do not take advantage of the decentralized nature of the blockchain. Businesses with a centralized structure do not need to execute costly consensus algorithms to keep track of activities that happen within. The intriguing property of the blockchain is that it allows the creation of immutable records that cannot be modified after the fact by anyone. The combination of the blockchain structure with serial proofs of work ensures that we can create an immutable ledger, regardless of the way it is created.

Therefore, we present a new setting in which each participant maintains a personal blockchain with pointers to other participant's chains. Each participant can use his chain however he sees fit. We expect them to use it as a ledger of records that they wish to share (be it publicly or privately), focusing on the fact that they were created sometime in the past. We will call this setting a *web of trust* but, in contrast to other uses of the term[1], the trust is gained through proofs of work instead of trust in one particular actor. If a blockchain is shown to be immutable and well constructed, outside parties can be certain that things in the blockchain cannot be changed in the future. This property is desirable in itself, but blockchains can provide even more. Blockchains supported by proofs of work ensure that some work must be invested to create them. This work can be translated as time under the right conditions. Thus, a blockchain can serve as a timestamping mechanism, as the time invested in the creation of blocks can act as a proof that something happened in the past. This possibility has

[1]Like in PGP.

been explored in [GMG15] and services exist already to add timestamps to the Bitcoin blockchain. However, the Nakamoto model cannot be used in a setting where chains are maintained by individual agents, as it would require a considerably high difficulty parameter to be considered trustworthy, and that implies a tremendous waste of energy. A fundamental motivation for the Nakamoto protocol is consensus, which in some way *justifies* the energy waste. Without the need of agreement in a personal chain, the energy waste inherent in the Nakamoto protocol is too high for practical use. However, the investment of time in serial proofs of work is a more direct way to create guarantees of the passage of time.

Blockchains maintained in the style of SingleLipwig in Chapter 3 can serve as personal time-stamping mechanisms[2], where trust is handled by the serial proofs of work. However, this setting is quite weak, as it is based on too many assumptions: that the participant will continuously maintain the chain and that the rate of computation of the proofs of work is considerably fast. Therefore, to be able to effectively use this model we will need an additional structure. The solution is simple: building a network of individual chains which compound each other's security. Every individual chain contains pointers to the blocks of other chains. Therefore, it becomes possible to verify when one chain changes, as pointers to the original chain exist in multiple other chains. Of course, the other chains are not intrinsically trustworthy either. However, if the network is sufficiently connected and there is at least one chain that can be fully trusted, it is possible to create trust in the network as a whole. The trusted chain can *lend* its trust to the other chains by maintaining pointers to the other chains. If a chain corresponds to the pointers found in another chain, then it can be considered at least as trustworthy as the chain that points to it. This relationship is transitive, meaning that the whole network can gain security from a single chain, even if there are no direct connections. Note that if the rate of one PoW chain is sufficiently high, this is reason enough to trust it. No more trust assumptions are needed.

The idea of blockchains *borrowing* security from more trusted chains is not a new one. In [BCD$^+$14], the authors present a system where *sidechains* can be built on top of the Bitcoin blockchain and therefore inherit the trust from Bitcoin. While that work does things that go beyond what we are trying to do, like transferring assets between chains, they depend on a pre-existing trustworthy chain. The paradigm we present here is self sustainable, as the trust does not need to extend beyond trust in the serial proofs of work. In practice, if there is an entity that is trusted outside the network, the connections to it can be deemed trustworthy. The important fact is that this pre-existent trust is not necessary if the blockchain is maintained properly and a chain is of the correct strength.

[2]We could also use proofs of variable strength like in SingleVarLipwig, but for simplicity we will focus only on SingleLipwig.

A setting like the one we are presenting gives rise to many adversaries with distinct goals. One adversary might be interested in presenting different valid chains. Another one might be interested in forking a particular chain or in isolating a chain to remove the immutability guarantees that it acquires from the network. These adversaries can have divergent and even opposing goals. Furthermore, it can be undesirable for them to undermine the integrity and security of the network. The adversary can be incentivised to not disrupt the functioning of the network too strongly, as it will bring negative consequences to her too.

6.1 Example

We present an example with three participants: Havelock, Moist and Cosmo ($\mathcal{H}$, $\mathcal{M}$ and $\mathcal{C}$ respectively) and an outside observer. Havelock has access to the most powerful processor and can create proofs of work at the highest rate. He also runs his chain with no interruption and always using the strongest proofs of work possible. Moist, on the other hand, has a mediocre processor that runs only at half the rate of Havelock's and occasionally stops computing his chain. Moist wants to prove to an outside observer that at certain point in the past, he had registered something in a block in his chain, which at this point is k blocks deep. Cosmo has managed to get a hold of a processor as strong as Havelock's as well as Havelock's and Moist's secret keys. His only goal is to stop the outside observer from believing Moist. This outside observer has knowledge of the functioning of the blockchain but does not trust any of the participants.

Let $BC_{\mathcal{H}}$, $BC_{\mathcal{M}}$ and $BC_{\mathcal{C}}$ be Havelock's, Moist's and Cosmo's blockchains respectively. For simplicity, we assume that there is a rigid round structure, which means that blocks are added at regular intervals. However, a participant may choose not do add a block for a round, which is the case for Moist but not for Havelock. Thus, at the point of time in which Cosmo acquires the secret keys of the other participants we have that $\mathsf{len}(BC_{\mathcal{H}}) = i$ and $\mathsf{len}(BC_{\mathcal{M}}) = j$. Cosmo is interested in forking Moist's chain from block $BC_{\mathcal{M}}[j-k]$. When the outside observer asks for Moist's chain, Cosmo may cut it at $BC_{\mathcal{M}}[j-k-1]$ and hand that to the observer. Because the observer knows that Moist does not necessarily add a new block at every round, he has no way of knowing the real length of Moist's chain, therefore he considers $BC_{\mathcal{M}}[0, j-k)$ to be a valid chain. However, Moist could present his (longer) chain which contains $BC_{\mathcal{M}}[j-k]$. Because the proofs are half as strong as the ones found in $BC_{\mathcal{H}}$, the observer only has assurance that the block was created $k/2$ rounds ago, which might not be enough. More importantly because we have that $\gamma_{\mathcal{C}} = \gamma_{\mathcal{H}} = 2\gamma_{\mathcal{M}}$, for every block that Moist can add to his chain, Cosmo can add two. Therefore, with enough prevision, Cosmo may create enough blocks such that the chain with $BC^*_{\mathcal{M}}[j-k]$ instead of $BC_{\mathcal{M}}[j-k]$ has stronger security.

Therefore, the model itself does not offer many security guarantees. Cosmo has enough opportunities to modify Moist's chain because of the difference in rate. There is a simple way to change this. Whenever Moist creates a block, he sends it to Havelock, who will add a pointer to it in his next block. In this case, when the outside observer arrives, he checks Havelock's chain as well as Moist's. Because all of Moist's blocks are reflected in Havelock's chain, the observer knows the length of Moist's chain. In addition, Havelock's chain always adds a block and it has the fastest possible rate, so the observer knows how much time has passed since a certain block of $BC_{\mathcal{H}}$ was issued. This serves as a lower bound for the time that has elapsed since any of Moist's blocks were created, as they are all reflected on Havelock's chain.

Now, if Cosmo wishes to change Moist's chain, he must not only change that one chain but also Havelock's. Fortunately for him, he has access to Havelock's secret key. Unfortunately for him, this will not be enough to change the chains. While Moist's chain is irregular in the creation of blocks, and can therefore be of any length, Havelock's chain must grow as time passes. Therefore, when Cosmo tries to change it, it grows at the same rate. Very similar to what we showed in Section 4.4, Cosmo will not be able to modify Havelock's personal chain and therefore will not be able to create a chain for Moist that an outsider will believe, as it will not coincide with what is encoded in Havelock's chain. Therefore, while Moist's rate is half of Havelock's, by connecting with Havelock's chain, Moist inherits the security from a stronger rate. Following this principle, it is possible to create a web of interconnected blockchains, each of them lending its security to the others and enforcing the trust on the system.

6.2 Beyond

We have presented a very simplified example of the model, but it can be enriched in many ways. Firstly, the example is based on one *ideal* blockchain that has the strongest proofs of work and is always running. Having one such chain to which everything is directly connected is not a practical possibility. However, the immutability guarantees that it provides, in particular the lower bound on the time in which the block has been created, can be propagated by indirect communication. In our example, if a new blockchain was created with pointers recorded in Moist's chain but not in Havelock's, this new chain would still inherit the security from the stronger chain. The security in the block in Moist's chain which points to the new chain is bolstered by the block in Havelock's chain which points to it. Therefore, trust can be transferred through chains.

Another notion that is important to take into account is the idea of which chains are connected with which others. Having a single trusted chain and looking for the shortest route towards it is not the way it will be presented in practice. For starters, it creates a centralising agent, when one of the major strengths of blockchain is the ability of avoiding these structures. On the other hand, on a network where connections are evenly distributed between all the participants, the strongest proofs of work at any given time are the ones which provide security to the web. An additional advantage of the network structure is that if someone is interested in attacking one chain, he must attack all the others as well. Therefore, it is not only the strongest blockchain that defends the network from the adversary, but all the chains that are a part of it.

While the presentation of this setting has focused on the security provided by serial proofs of work, a security intrinsically linked with clock time, there is no restriction that would dictate that all the chains in the network should be based in serial proofs of work. The serial proof of work paradigm is particularly adapted to this role, because it provides a deterministic guarantee that a certain time has elapsed from the creation of the block. This fact does not mean, however, that blockchains with other structures cannot be part of the network as well. While we focus on individual blockchains for the serial proof of work, distributed blockchains could also take part in the network. This also strengthens the network, as an adversary would now have to attack different strategies to bring down the network.

Conclusion 7

To conclude, we will summarise the main results of this thesis and discuss ideas for future work.

7.1 Summary of Results

The first part of this thesis consists of a survey of the current research in blockchain. It starts with a presentation of Bitcoin: an explanation of its history and its impact. After this, we mention the academic work that has been done in order to study it formally, as well as the weaknesses that have been found. We continue by mentioning the proposals for solutions for these issues, focusing on the academic perspective but mentioning the most important examples in practice. After this, we postulate that while permissioned blockchain protocols exist, none of them have similar immutability guarantees to Bitcoin. We follow this up by presenting a function that will provide the immutability guarantees for the blockchains in this thesis. We present the properties of modular square roots and its advantages. Finally, we briefly present the universal composability framework that will be used to construct the protocols in Chapters 4 and 5.

The following chapter presents the concept of a proof-of-work blockchain, PoW chain for short, through a couple of simple protocols. We define the structure of these chains as well as the two notions of security we are interested in. Security is based in clock time; the first notion quantifies the time needed to fork a blockchain and the second shows that a block in a chain must have been created at least some time in the past. We describe a prover-verifier game where the agent running the protocol attempts to fork the blockchain without a verifier noticing that this is happening. In the first protocol, a block is added at a fixed interval every time. We show that the prover can only fork the blockchain under certain assumptions and that even then, she cannot do so arbitrarily. In the second protocol, the prover is allowed to choose when to add a block, but that time must then be represented in the blockchain. In this case we show that security still holds as long as the verifier has a record of when each block was output.

The next chapter is concerned with the creation of the $\mathsf{IdealLipwig}^\tau$ protocol. First, we define the concept of a ledger chain, which is maintained by the whole network instead of individually, and we show how it can be connected with the PoW chains that are maintained by each participant. After this, we define the rules that each participant has to validate blocks, transactions and proofs of work. Additionally, we present a way to find participants that try to create forks and a mechanism to deal with them. After this, we present the building blocks of our protocol's consensus mechanism, BFT and $\mathsf{Consensus}$, and prove that they are secure under certain circumstances. We then build $\mathsf{IdealLipwig}^\tau$, ensuring that the circumstances for the correct functioning of the consensus are met, and we prove that it is secure in the classical blockchain protocol sense. After this, we show that the immutability results from the previous chapter also apply to all the blockchains in the protocol.

We then construct our main protocol, Lipwig^ω, by avoiding many of the idealising assumptions needed to make $\mathsf{IdealLipwig}^\tau$ work. In Lipwig^ω we do not assume perfect communication or strict rounds, yet the protocol's structure is similar to its predecessor. We use the same building blocks that we used in the previous chapter, showing that the security guarantees continue to hold in this new setting. We then prove that the immutability guarantees are still met in this setting. Additionally, we provide context to the proofs in Chapter 3, showing that constructions that seemed unnatural there are actually realised in this protocol. We follow this by briefly explaining the possibility of generating randomness over the blockchain, without the need of any additional assumptions. In the end, we present certain ideas to implement this protocol that are in line with the current concerns for implementations of blockchains.

In the last chapter, we present a new blockchain paradigm where PoW chains are maintained individually by participants but are interconnected by hash pointers. Instead of a network of agents agreeing over a single chain, we portray a web of individual blockchains that secure each other providing a web of trust between peers. We present a simple example of how this system would work and hint at the possibilities that this paradigm can have.

7.2 Further Research

In a field of study such as blockchain, implementation is a fundamental part of the work that must be done. There are multiple protocols in this thesis, but they all come from the same base. First, we would want to create a system to create and maintain our PoW chains. Having built this, there are numerous roads we can take. First, we could create a direct implementation of Lipwig^ω, with the goal of using it as the architecture of the so-called *consortium* blockchain. When following this road, we should take advantage of the specifications of our protocol to build a system that is modular, in order for it to be adaptable to different settings and consensus protocols. Another interesting direction would be to enhance existing blockchains with serial proofs of work, especially ones with modular structures, like Hyperledger Fabric. Enhancing existing blockchains with serial proofs of work can increase their security and also their functionality, as they could serve as randomness generators. Finally, we could build the necessary infrastructure to realise the web of trust we presented in Chapter 6, creating not only the blockchains but also a way to efficiently traverse the web.

On the theoretical side, the protocol and its components can be improved to be more flexible and secure. We present a proof-of-work function with certain desirable characteristics, but it is not the only one of its kind. We would want to have a function that can provide additional security guarantees, like quantum-safety, or one where the complexity gap between computation and verification is larger. The time difference in computing a modular square root and squaring a number is only linear, which we would like to improve. In [MMV11] some issues are raised with number-theoretic-based slow functions and [MMV13] presents an example of a slow function based on DAGs that might avoid some of these issues. This function is also of interest because verification takes logarithmic time over the size of the outputs. There are also some aspects of the protocol that we would like to improve in order to allow for robustness in an asynchronous setting. The current protocol is too rigid to work properly in such a setting, but we believe it could be modified to account for asynchronous communication and participants disconnecting and connecting from the protocol.

A natural question for a blockchain like this one is whether it can be expanded into a permissionless setting, the natural habitat of blockchains. Due to the fact that we are based on serial proofs of work, we already have a Sybil protection built into our model. This, however, affects our cheating prevention mechanism which becomes useless. Other parts of our protocol are based on the participants knowing the identity of all other participants in the network, which must be modified to continue to work. We believe that the inherent ability of the serial proofs of work to generate common randomness might lead to a successful blockchain protocol. Another interesting challenge would be the design of an incentive for participants to invest computational power and create blocks.

However, we believe the true permissionless setting for serial-proof-of-work based blockchains is the one presented in Chapter 6. While we have only presented the model broadly, there is still work to be done to prove the security and advantages of this paradigm. This is an interesting challenge as it presents a setting with multiple adversaries with different goals in mind. We believe that serial proofs of work in this setting will allow people to maintain a secure blockchain that can be used for anything from timestamping events and documents to building an infrastructure of trust in the web. This implies numerous challenges ranging from purely practical to highly theoretical, making it a hopefully very fruitful area of study.

Bibliography

[Bac01] Adam Back. Hash cash: A partial hash collision based postage scheme. *URL http://www. hashcash. org*, 2001.

[BCD$^+$14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *https: // www. blockstream. com/ sidechains. pdf* , 2014.

[BGM16] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *International Conference on Financial Cryptography and Data Security*, pages 142–157. Springer, 2016.

[BHS93] Dave Bayer, Stuart Haber, and W Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. *Sequences II: Methods in Communication, Security and Computer Science*, pages 329–334, 1993.

[Bit17] `http://realtimebitcoin.info/`, 9 August 2017.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Annual International Cryptology Conference*, pages 324–356. Springer, 2017.

[But17] Vitalik Buterin. Proof of stake faq. `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`, 10 July 2017.

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

[Can16] Ran Canetti. Universally composable security: A tutorial. https://www.youtube.com/playlist?list=PLqc9MPlwib9nSuyH4oUIwPsyDiZ4bwuEE, March 18 2016.

[CDE$^+$16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography Conference*, pages 61–85. Springer, 2007.

[CL$^+$99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[DPS16] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. http://eprint.iacr.org/2016/919.

[EGSVR16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *NSDI*, pages 45–59, 2016.

[ene17] Bitcoin energy consumption index, 25 September 2017. https://digiconomist.net/bitcoin-energy-consumption.

[ES14a] Ittay Eyal and Emin Gün Sirer. It's time for a hard bitcoin fork. http://hackingdistributed.com/2014/06/13/time-for-a-hard-bitcoin-fork/, 2014.

[ES14b] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[Eth16] What is ethereum? http://ethdocs.org/en/latest/introduction/what-is-ethereum.html, 2016. 41fc2c03.

[GKL14] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765, 2014. http://eprint.iacr.org/2014/765.

[GKL17] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *Annual International Cryptology Conference*, pages 291–323. Springer, 2017.

[GKP17] Juan A. Garay, Aggelos Kiayias, and Giorgos Panagiotakos. Proofs of work for blockchain protocols. Cryptology ePrint Archive, Report 2017/775, 2017. `http://eprint.iacr.org/2017/775`.

[GMG15] Bela Gipp, Norman Meuschke, and André Gernandt. Trusted timestamping using the crypto currency bitcoin. *iConference 2015 Proceedings*, 2015.

[Gro97] Lov K Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical review letters*, 79(2):325, 1997.

[Hyp17] Welcome to hyperledger fabric. `https://hyperledger-fabric.readthedocs.io/en/latest/`, 2017. 9e4f3b95.

[JM13] Yves Igor Jerschow and Martin Mauve. Modular square root puzzles: Design of non-parallelizable and non-interactive client puzzles. *Computers & Security*, 35:25–36, 2013.

[KJG$^+$16] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, 2016.

[KKJG$^+$17] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. Cryptology ePrint Archive, Report 2017/406, 2017. `http://eprint.iacr.org/2017/406`.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

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

[LVCQ16] Shengyun Liu, Paolo Viotti, Christian Cachin, and Vivien Quéma. Xft: Practical fault tolerance beyond crashes. In *OSDI*. usenix, 2016.

[LW15] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. `http://eprint.iacr.org/2015/366`.

[Mer80] Ralph C Merkle. Protocols for public key cryptosystems. In *Security and Privacy, 1980 IEEE Symposium on*, pages 122–122. IEEE, 1980.

[Mic16] Silvio Micali. Algorand: The efficient and democratic ledger. *arXiv preprint arXiv:1607.01341*, 2016.

[MKKS15] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 680–691. ACM, 2015.

[MMV11] Mohammad Mahmoody, Tal Moran, and Salil P Vadhan. Time-lock puzzles in the random oracle model. In *CRYPTO*, volume 6841, pages 39–50. Springer, 2011.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. Publicly verifiable proofs of sequential work. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 373–388. ACM, 2013.

[MXC$^+$16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[NC17] Arvind Narayanan and Jeremy Clark. Bitcoin's academic pedigree. *acmqueue*, 15-4, 2017.

[Oku05] Michael Okun. *Distributed Computing Among Unacquainted Processors in the Presence of Byzantine Failures.* Hebrew University of Jerusalem, 2005.

[Pev17] Ignotious Peverell. Introduction to mimblewimble and grin. `https://github.com/ignopeverell/grin/blob/master/doc/intro.md`, 2017. 55eb2f6.

[Poe14] Andrew Poelstra. Distributed consensus from proof of stake is impossible, 2014.

[Pra07] Terry Pratchett. *Making Money.* Doubleday, 2007.

[PS16a] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. Cryptology ePrint Archive, Report 2016/916, 2016. `http://eprint.iacr.org/2016/916`.

[PS16b] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. Cryptology ePrint Archive, Report 2016/917, 2016. `http://eprint.iacr.org/2016/917`.

[PS16c] Rafael Pass and Elaine Shi. The sleepy model of consensus. Cryptology ePrint Archive, Report 2016/918, 2016. `http://eprint.iacr.org/2016/918`.

[PSas16] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Report 2016/454, 2016. `http://eprint.iacr.org/2016/454`.

[PSL80] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[PW16] Cécile Pierrot and Benjamin Wesolowski. Malleability of the blockchain's entropy. In *ArcticCrypt 2016*, 2016.

[Sir17] Emin Gün Sirer. What could go wrong? when blockchains fail. Business of Blockchain, 2017.

[STV$^+$16] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities" honest or bust" with decentralized witness cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 526–545. Ieee, 2016.

[vS14] Nicolas van Saberhagen. Cryptonote v 2.0, 2014.

[Wil16] Zooko Wilcox. Hello, world! `https://z.cash/blog/helloworld.html`, 2016.