Levels of Decentralization and Trust in Cryptocurrencies: Consensus, Governance and Applications

Sarah Azouvi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London

2021

I, Sarah Azouvi, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Since the apparition of Bitcoin, decentralization has become an ideal praised almost religiously. Indeed, removing the need for a central authority prevents many forms of abuse that could be performed by a trusted third party, especially when there are no transparency and accountability mechanisms in place. Decentralization is however a very subtle concept that has limits.

In this thesis, we look at the decentralization of blockchains at three different levels. First we look at the consensus protocol, which is the heart of any decentralized system. The Nakamoto protocol, used by Bitcoin, has been shown to induce centralization through the shift to mining pools. Additionally, it is heavily criticized for the enormous amount of energy it requires. We propose a protocol, Fantômette, that incorporates incentives at its core and that consumes much less energy than Bitcoin and other proof-of-work based cryptocurrencies.

If the consensus protocol makes it possible to decentralize the enforcement of rules in a cryptocurrency, there is still the question of who decides on the rules. Indeed, if a central authority is able to determine what those rules are then the fact that they are enforced in a decentralized way does not make it a decentralized system. We study the governance structure of Bitcoin and Ethereum by making measurements of their GitHub repositories and providing quantitative ways to compare their level of centralization by using appropriate metrics based on centrality measures.

Finally, many applications are now built on top of blockchains. These can also induce or straightforwardly lead to centralization, for example by requiring that users register their identities to comply with regulations. We show how identities can be registered on blockchains in a decentralized and privacy-preserving way.

Impact Statement

Decentralized cryptocurrencies have the potential to disrupt financial services and give a real financial alternative to traditional institutions, especially for those who live under abusive governments. The results in this thesis could meaningfully help both industry and academia when designing decentralized cryptocurrencies.

First, in a world where climate change is the main challenge faced by future generations, our Fantômette protocol could be used as an alternative to the energy consuming Proof-of-Work. Additionally, our protocol was the first one to be argued $\varepsilon-(k,t)-$robust (under the adversary specifications that we detail in this thesis). This type of property together with our research on game theory and security has the potential to impact the way we reason about the security of blockchains.

Our work on governance was the first of its kind and has already garnered interest from the community as it was reported on one of the major cryptocurrency media at that time, Coindesk [1]. It thus has the potential to generate a new line of research around the quantification of centralization that could have a real impact on how governance structures are designed, as it would help measure their level of openness and centralization.

Lastly, our work on identity registration could help make blockchains more mainstream as, in order to comply with regulations, many services will have to ask for more information about users. We present a way to do this without compromising the privacy of users.

Finally, the content of this thesis has been presented on multiple occasions, both at academic venues and industry conferences. It is also available freely online.

Acknowledgements

I am so grateful I had the opportunity to work on such a fascinating topic that I believe could have a real positive impact on society. This would not have been possible without the support of many people. Sarah Meiklejohn who took me as her PhD student, introduced me to exciting research problems, and gave me the freedom to pursue my interests. George Danezis, my second supervisor who provided general guidance and before that supervised my MRes, along with Gianluca Stringhini. Both contributed to my reconversion into information security. Philip Treleaven who funded my PhD and supported my change of topic in the early stage of my MRes.

I am grateful for my co-authors who contributed to the work in this thesis: Mustafa Al-Bassam, Shehar Bano, Alexander Hicks, Mary Maller, Patrick McCorry, Steven Murdoch Alberto Sonnino.

I cannot thank all of my office mates in 6.22 enough. They made this PhD experience enjoyable and gave me an academic family, and everyone at UCL Pole Fitness Society who provided me a space to relax and have fun outside of work. This would not have been the same without you.

I am also grateful to Henri Stern and all the Protocol Labs team and Evan Cheng and the Calibra team for exciting collaborations.

Finally, I would like to thank all my friends and family, especially my mother, my brother, and my cousin Eva for their continuous love and support. Lastly, Alex, my partner in all aspects of my life, one sentence would not be enough to thank you for your love and support. You push me to be a better person.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Connaître ce n'est pas démontrer, ni expliquer. C'est accéder à la vision. Mais pour voir, il convient d'abord de participer.
Antoine de Saint-Exupéry - Pilote de guerre (1942)

1.1 Problem statement

Motivated by a rejection of any central authority, Bitcoin was created by the pseudonymous Satoshi Nakamoto [2]. It is a digital currency, not regulated by any central bank or government, which relies on a peer-to-peer network to function. In this system, no one entity can act to censor transactions or prevent individuals from joining the network (as is possible with traditional institutions [3]). Decentralization has long been some sort of ideology to cipherpunks and crypto-anarchists; whether it comes to financial freedom (e.g., Bitcoin), privacy and anonymity (e.g., Tor) or data sharing (e.g., BitTorrent). Indeed such decentralized systems help defend human rights around the world [3, 4, 5], especially to those under oppressive governments. The ideal decentralized system does not depend in any way on any single party. Troncoso et al. [6] give an overview of decentralized systems, defining a decentralized system as "a distributed system in which multiple authorities control different components and no single authority is fully trusted by all others". This highlights the fact that every component of the system should be decentralized. This can be hard to achieve in practice, and the level of decentralization of a system should always be looked at critically. For example, a single authority

controlling a distributed system, whose components are running on different machines, is not decentralized. Similarly, a decentralized system where all important parties are independent but under the jurisdiction of a single government may not truly be decentralized. All these independent parties may also depend on a very few hardware manufacturers (or other service providers). A central component of a decentralized system is a *consensus protocol*, by which the system's participants can agree on its current state and use that information to take various actions. In the case of Bitcoin, this means appending transactions to a ledger, also known as *mining*. Consensus protocols have been studied for decades in the distributed systems literature, and classical protocols such as Paxos [7] and PBFT [8] have emerged as "gold standards" of sorts, in terms of their ability to guarantee the crucial properties of safety and liveness even in the face of faulty or malicious nodes.

Bitcoin has renewed interest in consensus protocols, due largely to two crucial new requirements: openness and incentivization. First, classical consensus protocols were designed for a closed and relatively small set of participants, whereas in an open (or "permissionless") setting the goal is to enable anyone to join. This requires the design of new consensus protocols that can both scale to handle a far greater number of participants, and also ones that can address the question of Sybil attacks [9], as participants may no longer be well identified. Bitcoin solves this by using a consensus protocol sometimes referred to as Nakamoto protocol or Proof-of-Work (PoW) where, in order to gain the right to enter new information in the system, a participant must solve a computational puzzle. In theory, each peer in the network has an amount of "power" proportional to their computing capabilities. Provided a majority of the computational power is honest, only valid information enters the ledger. The underlying feature of this system is a globally visible, append-only ledger or *blockchain*.

This has in fact been proved secure [10, 11], but provides Sybil resistance by requiring enormous amounts of computational power to be expended. As such, it has also been heavily criticized for the large amount of electricity that it uses. Furthermore, it has other subtle limitations; e.g., it does not achieve any notion

of fully deterministic finality and some attacks have been found on its incentive scheme [12, 13].

With its rise in popularity, the price of Bitcoin has increased exponentially, and the price of mining has followed this trend with the apparition of ASIC mining hardware. As a result, a phenomenon known as mining pools has appeared in which miners join their resources to mine, together, more blocks. This goes against the decentralized ideal originally envisioned. If few miners were to control most of the system, they could collude and abuse the system and its users easily by, for example, censoring some transactions or other miners.

Additionally, there is the question of who makes the rules in the first place. Even if the enforcement of the rules is decentralized through the consensus protocol, if a central authority was able to decide on these rules the system would not be truly decentralized. For example, in the case of Bitcoin, its pseudonymous creator Satoshi Nakamoto made many arbitrary decisions: he decided the time interval between block generation, the mining reward, or the size of the blocks. These are very important decisions that affect the security and performance of Bitcoin and have generated heated arguments. Especially after Satoshi Nakamoto withdrew from the project, it was not clear who could have the authority to make important decisions. There is indeed a governance structure in all blockchains, and these governance structures have a seemingly inherent degree of centralization. Most cryptocurrencies address this by open sourcing their code and opening their protocols to so-called "improvement proposals," in which anyone can propose changes to the high-level protocol. These improvement proposals serve not only to diminish the degree of centralization in the maintenance of the platform, but also provide a significant degree of transparency into the decision-making process. In established platforms, however, it is not clear what punishment is possible if the core developers misbehave. Ultimately users might not have any choice but to accept their decision. The following is thus an important open question: in cryptocurrencies, where participants typically feel a strong sense of ownership, to what extent are existing platforms open and transparent? A centralized and opaque governance

structure could lead to abuse and loss of trust.

Finally, blockchains have received a lot of attention for their potential applications. In addition to being used as the underlying architecture for cryptocurrencies, they have been discussed for achieving decentralized versions of identity management, [1] DNS and public-key infrastructures, [2] notary publics,[3] and file storage.[4] While centralized versions of these systems already exist, the attraction of distributed ledgers is that they minimize the extent to which users must place trust in a single entity such as a certificate authority. For some of these applications, it may be necessary to know some information about their users, e.g., gambling services would like to know that their users are over 18. In all existing deployments of distributed ledgers, users identify themselves using *pseudonyms* — or even more anonymous identifiers, as in the cryptocurrency Zcash [14] — that they create themselves. The use of pseudonyms is important for two reasons: first, all existing distributed ledgers are *transparent*, meaning their contents are globally visible, so having users reveal their real-world identities would completely violate their privacy. Second, allowing users to generate their own identifiers is necessary to preserve the openness of the system and allow anyone to join. While these "on-chain" pseudonyms are thus seemingly quite useful (and to some extent necessary) in public distributed ledgers, they make it challenging to associate the necessary information about users without compromising their privacy. Ideally, users should not be forced to reveal to anyone the tie between their real-world identity and their pseudonym(s), as this compromises their entire privacy. The question is then how to provide a way of registering certain attributes without undermining the privacy of users?

As argued above, a decentralized system requires all of its components to be decentralized, at least to some extent. This thesis tackles the outlined problems of decentralization within blockchains at three different levels. First, at the protocol

[1] `https://onename.com`
[2] `https://namecoin.info`
[3] `https://proofofexistence.com/`
[4] `https://filecoin.io`

level by proposing a new consensus protocol that provides a lower barrier to entry. Second, at the governance level by quantifying the centralization in the governance structure of decentralized blockchains. Third and finally, at the application layer by proposing decentralized registration systems that maintain privacy.

We present in Chapter 4 a new consensus protocol, Fantômette based on proof-of-stake (PoS). The idea behind proof-of-stake is to offer lower energy consumption than the current protocol by using *stake* (i.e. the amount of coins that one owns in the currency) instead of using proof-of-work to handle Sybils. Our protocol thus solves the problem of how to enforce the rules of a cryptocurrency in a decentralized manner without relying on intensive energy consumption. Additionally, proof-of-stake does not require any investment in hardware and thus lowers the barrier to entry, aiding the protocol to be executed in a more decentralized manner.

In Chapter 5, we quantify the level of centralization in the governance process of Bitcoin and Ethereum, two of the main cryptocurrencies, by making measurements on their Github repositories and providing a quantitative method of comparison.

Finally, in Chapter 6 we propose different decentralized registration processes for blockchains that maintain some notions of privacy and decentralization.

The outline of the rest of this thesis is as follows. In Chapter 2, we give some general background on the concepts discussed in this thesis as well as the notations that will be used. In Chapter 3, we present a literature review of the research most closely related to this thesis and discuss the conclusions from that literature review. We conclude in Chapter 7 and present some open problems in the field.

1.2 List of Papers

This section presents the author's published works, ordered chronologically. All papers are joint work.

Sarah Azouvi, Mustafa Al-Bassam, and Sarah Meiklejohn. Who am I? Secure identity registration on distributed ledgers. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 373–389. Springer, 2017

This paper proposes different decentralized registration protocols for public keys on distributed ledgers. The problem was introduced by Meiklejohn who suggested looking at a decentralized version of Casascius coins [16]. The author then designed the proposed protocols and associated threat model. Al-Bassam wrote the implementation. We present this work in Chapter 6.

Sarah Azouvi, Mary Maller, and Sarah Meiklejohn. Egalitarian Society or Benevolent Dictatorship: The State of Cryptocurrency Governance. In *22nd International Conference on Financial Cryptography and Data Security*, 2018

This paper proposes quantitative methods to analyze the level of decentralization in the governance of Bitcoin and Ethereum. The metrics used were discussed by all the authors. The author scraped the data and computed all the graphs and measures. This work is presented in Chapter 5.

Sarah Azouvi, Alexander Hicks, and Steven J Murdoch. Incentives in Security Protocols. In *Cambridge International Workshop on Security Protocols*, pages 132–141. Springer, 2018

This position paper argues that incentives are paramount in security protocols and should be included in every stage of their development. The authors illustrate this argument with three examples: EMV (card payments), Tor and cryptocurrencies. The idea originated from a discussion between Hicks and the author regarding Tor and cryptocurrencies. Murdoch then suggested the example of EMV. Hicks wrote the part about Tor, Murdoch about EMV and the author about cryptocurrencies. The part that the author wrote is used in Chapter 3.

Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. In *Advances in Financial Technologies*, 2019

This paper is a systemization of knowledge of blockchain consensus protocols. The idea emerged from a reading group organized weekly by Bano in which all the authors participated. The author suggested the chronological approach of considering first traditional consensus protocols as they were designed before Bitcoin and comparing them to the newer blockchain consensus protocols. Bano took all the initiatives regarding the outcome of the paper. The author wrote the proof-of-stake part as well as the part on incentives. The part that the author wrote is used in Chapter 3.

Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Betting on blockchain consensus with fantomette. *arXiv preprint arXiv:1805.06786*, 2018

This paper proposes a new proof-of-stake consensus protocol. The leader election protocol Caucus and its modelization were designed by Meiklejohn and the author, the Ethereum implementation of a previous version of the leader election [21] was designed by McCorry. The Fantômette protocol was designed by the author who also proposed the associated security model and the proofs and performed the simulations. This work is presented in Chapter 4.

Sarah Azouvi and Alexandre Hicks. SoK: Tools for Game Theoretic Models of Security for Cryptocurrencies. *arXiv preprint arXiv:1905.08595*, 2019

This paper is a systemization of knowledge of the work that bridges the fields of computer science and game theory; specifically looking at models that could be used when designing blockchains protocols. The author came up with the idea of the paper. The outline of the paper was discussed between both authors. The author wrote most of the paper. Part of this work is used in Chapter 2.

Chapter 2

Background Definitions and Notation

In this chapter, we give some background on the concepts that will be discussed in this thesis. We start by introducing the necessary cryptographic primitives and notations. Then in Section 2.2 we give some general background on blockchains, including on proof-of-stake. Some parts of this chapter are based on [22].

2.1 Cryptographic Primitives and Notation

Following standard cryptographic notation, we use $x \xleftarrow{\$} S$ to denote the process of sampling a member uniformly from S and assigning it to x. In particular, we use $x \xleftarrow{\$} [n]$ to denote sampling x uniformly from $\{1,\ldots,n\}$. We use $y \leftarrow A(x_1,\ldots,x_n;R)$ to denote running algorithm A on inputs $x_1,\ldots,x_n$ and random coins R and assigning its output to y. By $y \xleftarrow{\$} A(x_1,\ldots,x_n)$ we denote $y \leftarrow A(x_1,\ldots,x_n;R)$ for R sampled uniformly at random. Algorithms are randomized unless explicitly noted otherwise. "PT" stands for "polynomial time."

We say that two distribution ensembles $\{X_n\}$ and $\{Y_n\}$ are computationally indistinguishable if for any non-uniform probabilistic polynomial time algorithm A: $|\Pr[A(X_n)=1]-\Pr[A(Y_n)=1]|$ is negligible; we denote this as $X \approx Y$.

2.1.1 Digital signatures

Both Bitcoin and Ethereum rely on ECDSA for signing. In what follows we use $(pk,sk) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$ to denote key generation, $\sigma \xleftarrow{\$} \mathsf{Sig.Sign}(sk,m)$ to denote signing, and $0/1 \leftarrow \mathsf{Sig.Verify}(pk,m,\sigma)$ to denote verification.

2.1.2 Hash functions

A hash function is a function $H : \{0,1\}^* \to \{0,1\}^\ell$; i.e., a function that maps strings of arbitrary length to strings of some fixed length ℓ. When a hash function is modeled as a random oracle, this means that computing $H(x)$ is modeled as (1) looking up x in some global map and using the value of $H(x)$ if it has been set already, and (2) if not, picking a random value $y \xleftarrow{\$} \{0,1\}^\ell$ and setting $H(x) \leftarrow y$ in the map.

There are many desired properties of hash functions that trivially hold when they are modeled as random oracles. For example, hash functions satisfy *pre-image resistance* meaning that given a value h, it is not feasible except with negligible probability to find a value x such that $H(x) = h$. Hash functions are also collision-resistant meaning that it is not computationally feasible to find two values x_1 and x_2 such that $H(x_1) = H(x_2)$ and *second pre-image resistance* meaning that given x_1, it is not computationally feasible to find x_2 such that $H(x_1) = H(x_2)$.

2.2 Blockchains and Cryptocurrencies

2.2.1 Bitcoin and Ethereum

Bitcoin is a decentralized cryptocurrency, meaning that it is maintained by a network of peers rather than a central authority. In order to keep track of transactions and thus of the balance of accounts, Bitcoin uses a shared public ledger called blockchain. It is completely transparent meaning that everyone can have access to all the transactions history. Within the system, users are represented by *addresses* addr, each of which is uniquely linked to a pair of public and private ECDSA keys (pk, sk). This provides *pseudonymity* to users, meaning that their identities are not visible on the ledger, only their pseudonyms (i.e., their addresses). We denote by $\mathsf{addr}(pk)$ the address associated with pk. Every time Alice wants to pay Bob using Bitcoin she generates a *transaction* $\mathsf{tx}(\mathsf{addr}(pk_A) \to \mathsf{addr}(pk_B))$ and signs it with her private key sk_A. More generally, Bitcoin transactions can have arbitrarily many input and output addresses, in which case the transaction must be signed by all private keys associated with the input addresses, or even m-of-n multi-signature transactions, in which a transaction must be signed by the private keys associated

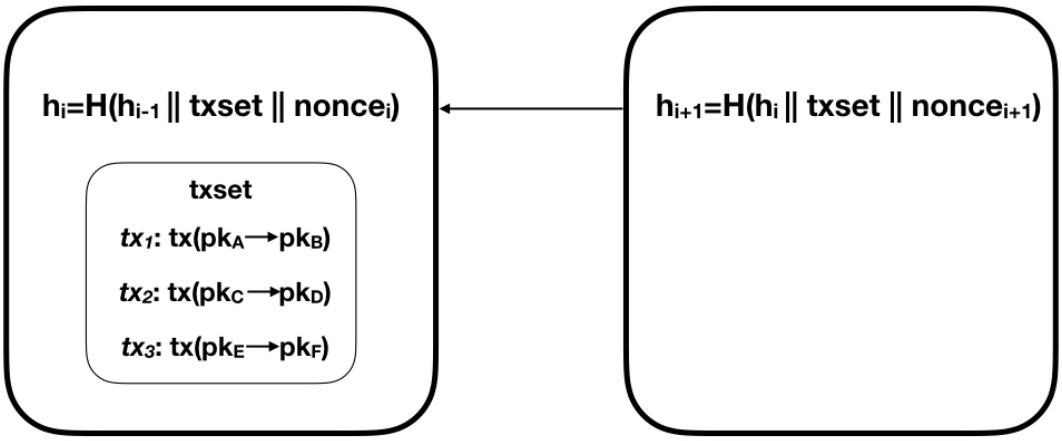


Figure 2.1: Block structure (simplified).

with at least m of the input addresses. Alice then broadcasts the signed transaction to the network, which checks its validity (i.e., that the amount of Bitcoin has not already been spent before and that the signature on the transaction is valid).

Transactions are then grouped into blocks that are chronologically ordered by pointing to the hash of the previous block, this forms a chain of blocks or *blockchain*. Due to this structure, changing one transaction in one block would require changing all the following blocks as this will change the hash for that block. For example, in Figure 2.1 changing tx_1 will result in a new hash h_i and thus in a new hash h_{i+1}. The blockchain is thus an append-only data structure.

In order for all the peers in the system to agree on their view of the blockchain (i.e., to avoid double-spending), they proceed in a consensus algorithm sometimes referred to as proof-of-work or Nakamoto Consensus. Each participant, called a *miner* tries to solve a moderately-hard computational puzzle: finding a nonce such that $h_i = H(h_{i-1}||\mathsf{txset}||\mathbf{nonce}) < \mathsf{target}$ where target is adjusted so that on average one block is found every ten minutes, i is the height of the block and h_{i-1} is the hash of the previous block. The first miner to solve this puzzle broadcasts its newly created block (which contains txset, h_{i-1} and **nonce**) to its peers. The other miners then check the validity of the block and add it to their blockchain. In the rare case where two miners find a block at the same time, we say that there exists a *fork* in the blockchain. Miners decide on which block to mine off of, Bitcoin then follows the *longest chain* rule, meaning that whichever chain creates the most blocks is the

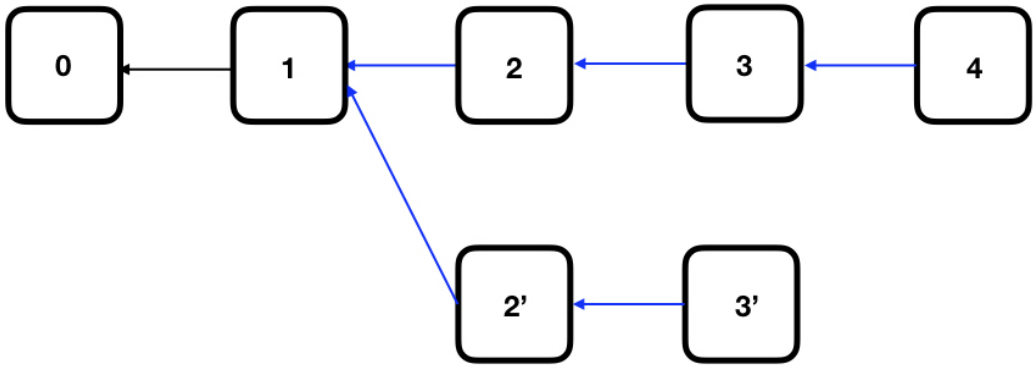


Figure 2.2: Blockchain fork.

one that is considered valid. Blocks that have not been included in that chain are thus abandoned. For example in Figure 2.2, the longest chain is {1,2,3,4}, meaning that blocks 2' and 3' are abandoned and will not be included in the blockchain.

In order to incentivize peers to join the network, Bitcoin has a reward scheme where the miner who gets to mine the block that is added to the blockchain receives an amount of bitcoins (12.5 bitcoins as of 2019) to compensate them for the (computational) work that they expend. They also receive optional transaction fees added by the users. The security of Bitcoin and other proof-of-work cryptocurrencies relies on the fact that it is costly to create a block due to the amount of energy required. An adversary would need to buy a lot of computational power in order to deliberately create a fork. This property gives to the blockchain most of its desirable qualities. First the blockchain is immutable, this is due to the price of creating blocks as well as their structure (linked by hash) explained above. If an adversary was to change the content of a block, it would need to change all of the following ones, incurring a large cost. This data structure hence provides full auditability as anyone with access to the blockchain can thus verify the whole history without fear of it being altered. This implies that nodes in the system need not trust each other.

Joining the protocol requires no registration of any sort, giving it its pure decentralized and open nature. The computational cost behind proof-of-work then solves the problem of Sybils where an adversary creates many identities to control the system. Decentralization and openness are one of the main properties that differentiate a blockchain consensus protocol from more traditional consensus protocols: Byzantine Fault Tolerant (BFT) consensus protocols [8]. In classical consensus

protocols, the set of participants needs to be known in advance and is closed, not everyone can decide to take part.

Bitcoin also allows users to program *smart contracts*, thanks to its scripting language, *Script*. A simple example of smart contracts is multisignature transactions, mentioned earlier, i.e., a transaction that requires m signatures, out of a set of n designated public keys. As another example, Bitcoin provides a *Timelock* smart contract primitive that prevents some coins from being used before a specified time in the future. Timelocks can then be used for the design of more complex functionalities, such a payment channels, that allow parties to exchange funds offline. The execution of a smart contract is enforced by the consensus protocol, just like the transactions. In order to prevent infinite loops, Bitcoin's script language is not Turing complete.

Ethereum is a project that works similarly to Bitcoin but provides a Turing-complete language, allowing for the definition of more complex smart contracts. For example popular Ethereum smart contracts include electronic voting or creation of new tokens (ERC20). In Ethereum, a smart contract consists of program code, a storage file, and an account balance. The program's code is executed by the network, which is responsible for maintaining a consistent view of the state of every contract in the blockchain. Users can call the contract by sending transactions to its address, which updates the state of the contract in the blockchain. Moreover, the execution of a program's instructions induces a cost; the currency used to pay for it is called gas. Ethereum transactions contain a destination address to (that can specify either the location of a stateful smart contract, or just a regular user-owned address), an amount amt to be sent (denominated in ether), an optional data field data, a *gas limit*, and a signature σ authorizing the transaction with respect to the sender's key-pair (pk, sk) (just as in Bitcoin). We denote the process of creating a transaction as (ignoring all but the data field) $\mathsf{tx} \xleftarrow{\$} \mathsf{FormTx}(sk, \mathsf{data})$, and the process of verifying a transaction as $0/1 \leftarrow \mathsf{VerifyTx}(\mathsf{tx})$.

2.2.2 Centralization

2.2.2.1 Mining Pools

Looking back at the initial success of Bitcoin, it has evolved to become different in many ways from the intended design and the idea of "one-CPU-one-vote" envisioned by Nakamoto. Because the price of mining has increased exponentially with the popularity of Bitcoin, miners have started forming mining pools, where they join their resources to mine, together, more blocks. The reason for doing so is that by pooling their resources miners will be able to mine more blocks, they will then share their gain accordingly, meaning that their average payoff is theoretically the same but their variance is reduced as they do not need to create a block in order to get some reward. Due to the depreciation of the hardware and the constant cost of electricity, this is desirable. Indeed with a high variance a "small" miner may have to wait too long before they receive a reward and may not be able to pay for their electricity before they do so. Mining hardware itself has changed, as ASICs have become more popular, and essential for profitable mining, increasing even more the barrier to entry to mining.

Pools are obviously a big threat to the security of cryptocurrencies as they could enable 51% attacks (where an adversary takes control of more than half of the computational power), which have already happened to other cryptocurrencies.

As of August 2019, the most important 51% attack has targeted Ethereum Classic, which is the 16th largest cryptocurrency by market capitalization [23]. Attacks have gradually targeted cryptocurrencies that are ranked higher on the list and it can be expected that this trend will continue, in particular when prices of such attacks are favored by market downturns.

2.2.2.2 Governance model

Cryptocurrencies have an inherent central component as someone has to come up with the rules in the first place. In this section we are concerned with who makes the rules in Bitcoin and Ethereum and especially how they are updated.

Bitcoin Bitcoin [2] was created by the pseudonymous Satoshi Nakamoto, who deployed the currency on January 3 2009. There has been no sign of him since 2011, when he said he would "move on to other things" and handed over control of the project to some of the members of the community who were prominent at that time. In September 2012, some important members of the community created the Bitcoin Foundation, a non-profit organization based on the model of the Linux Foundation, but today there is also a significant development effort by Blockstream [24], a for-profit company run by the core developers, which has attracted significant investment [25].

Ethereum Ethereum [26] was created by Vitalik Buterin, and launched on July 30 2015. Its initial development was done by the Ethereum Foundation, a Swiss non-profit, but today there is also a significant development effort by Parity [27], which is a for-profit company developing one of the main Ethereum clients.

Improvement Proposals (IP) No system is perfect, and cryptocurrency protocols sometimes need updating due to flaws or vulnerabilities. These changes can be fundamental and affect all users. To keep the improvement decision process open and fair, most cryptocurrencies have an *Improvement Proposal* system, where anyone can propose changes to the protocol and discuss existing proposals. If support exists for a proposal, it may be incorporated into the codebase via one of the core developers merging their pull request. There is no formal definition on how an improvement proposal is agreed upon [28]. The Improvement Proposals process is mainly happening on GitHub, however there are many other places for discussion, such as mailing lists, forums and IRC.

Chain Splits When disagreements occur in cryptocurrency communities, the only way to resolve them might be for the communities to split. Anyone disagreeing with the current core developers can fork the code and create their own currency. This has happened in both Ethereum and Bitcoin. In June 2016 more than 50M USD of ether were stolen due to a code vulnerability [29]. The Ethereum Foundation decided to roll-back in time in order to take the stolen ether back from the hacker. Arguing that this contradicts the fundamental immutability property of blockchains, some

members of the community forked Ethereum and Ethereum Classic was born [30]. The Bitcoin "block size" debate has been ongoing for years. Arguing that one of the main limitations of the Bitcoin protocol is scalability and that this problem could be solved with larger block sizes, some members of the community forked Bitcoin, resulting in Bitcoin Cash in August 2017 [31], as well as Bitcoin Gold in October 2017 [32]. Bitcoin Cash also split in November 2018, resulting in Bitcoin SV [33].

2.2.3 Proof-of-stake

By its nature, PoW consumes a lot of energy. Thus, some alternative consensus protocols have been proposed that are more cost-effective. As of this writing, arguably the most popular of these is called *proof-of-stake* (PoS) [34, 35, 36, 37, 38]. If we consider PoW to be a leader election protocol in which the leader (i.e., the miner with the valid block) is selected in proportion to their amount of computational power, then PoS can be seen as a leader election protocol in which the leader (i.e., the participant who is *eligible* to propose a new block) is selected in proportion to some "stake" they have in the system. This can be represented as the amount of coins they have (either in some form of escrow or just in total), or the age of their coins.

As security no longer stems from the fact that it is expensive to create a block, PoS poses several technical challenges [39]. The main three are as follows: first, the *nothing at stake* problem says that miners have no reason to not mine on top of every chain, since mining is costless, so it is more difficult to reach consensus.

Second, PoS allows for *grinding attacks*, in which once a miner is elected leader they privately iterate through many valid blocks (again, because mining is costless) in an attempt to find one that may give them an unfair advantage in the future (e.g., make them more likely to be elected leader).

Finally, in a *long range attack*, an attacker may bribe miners into selling their old private keys, which would allow them to rewrite the entire history of the blockchain.

PoS provides an arguably lower barrier to entry than PoW, as users only need to invest in the cryptocurrency. PoW requires hardware investment (ASICs), which

depreciates with time, as well as cooling devices, as it is no longer possible to mine with one's GPU due to the prevalence of mining pools. After the initial investment, a PoS protocol only requires one computer to be run, the electricity consumption is thus far lower.

Chapter 3

Literature Review and Related Work

In this chapter, we review the work that is related to this thesis. We present only the work that appeared before the first submission of this manuscript in August 2019. First in Section 3.1 we review the general work that exists on the (de)centralization of cryptocurrencies.When it comes to the decentralization of the enforcement of the rules (i.e., the consensus protocol), the literature is very diverse. The work that most closely resembles ours is the cryptographic literature on proof-of-stake. We evaluate and compare the state-of-the-art proposals alongside the requirements of scalability and incentivization in Section 3.2. As we will see, these protocols largely under-consider the game theoretic aspects of the problem, which motivated our work in Chapter 4. An interesting concept that has been introduced in the context of scalability of blockchains are blockDAGs, which our Fantômette protocol also uses. We review the work related to blockDAGs in Section 3.3. Then we discuss the incentive challenge linked to decentralized cryptocurrencies in Section 3.4, which motivated our security model in Chapter 4.

In Section 3.5, we review the existing literature on the governance of decentralized cryptocurrencies. As we will see there is not much work that has been done on the quantitative analysis of governance, which is what motivated our work in Chapter 5.

Finally, in Section 3.6 we review the literature on identity management on blockchains. Some parts of this chapter are based on [19, 22].

3.1 Blockchain Centralization

As previously argued, there are different layers where centralization can be observed (e.g. mining, governance). Much of the previous research examining decentralization on blockchains focuses specifically on Bitcoin, or on the general governance issues associated with blockchains. In terms of centralization within Bitcoin, Gervais et al. observe that some of the key operations in Bitcoin, in particular the mining process and the maintenance of the protocol, are not decentralized [40]. The centralization of mining has been empirically analyzed by Gencer et al. [41], who measured how decentralized Bitcoin and Ethereum's networks are. They found that three or four mining pools control more than half of the hash power of the network. This highlights the need for further research studying this occurrence of centralization and how decentralization can be maintained in practice.

Moore and Christin find a high degree of centralization in popular Bitcoin exchanges [42], and observe that popular exchanges are more likely to suffer security breaches. Böhme et al. look at the various centralized intermediaries within the broader Bitcoin ecosystem, such as currency exchanges, wallet providers, mixers, and mining pools [43]. They also evaluate the decisions that the designers make regarding how much money there should be in the system.

In terms of other cryptocurrencies, Reyes et al. examine the theft of 3.6 million ether from The DAO in June 2016, and discuss the lessons learned and the potential strengths and weaknesses of decentralized organizations [44]. Gandal and Halaburda analyze how network effects affect competition in the cryptocurrency market [45]. In particular, they look at the competition between different currencies and competition between cryptocurrency exchanges and observe that there was a "winner takes all" effect in early markets, but not today.

3.2 Proof of Stake Consensus Protocols

A number of proof-of-stake systems proposed in the cryptographic literature have provably secure protocols [46, 37, 47]. A common theme in these systems is to randomly elect a leader from among the participants via a lottery, who then appends

a block to the blockchain. Leader elections may be public, meaning that the outcome is visible to all the participants [37, 47]. Alternatively, in a private election, the participants use private information to check if they have been selected as the leader, which can be verified by all other participants using public information [46]. Leader elections based on private lotteries are resilient to DDoS attacks because participants privately check if they are elected before revealing it publicly in their blocks, at which point it is too late to attack them.

The nature of the lottery varies across different systems, but broadly it is either collaborative (i.e., it requires coordination between the participants) or independent. In Ouroboros [37], the participants run a multiparty coin-tossing protocol to agree on a random seed. The participants then feed this seed to a pseudo-random function defined by the protocol, that elects the leader from among the participants in proportion to their stake. In Ouroboros Praos [46] and Snow White [47], participants independently determine if they have been elected. Snow White selects participants for each epoch based on the previous state of the blockchain. Each of them independently checks if they have been elected as the leader. Snow White uses similar criteria for leader election as Bitcoin, i.e., finding a pre-image that produces a hash below some target. Participants are however limited to computing only one hash per time step (assuming access to a weakly synchronized clock) and the target takes into account each participant's amount of stake. In Ouroboros Praos, participants generate a random number using a verifiable random function (VRF). (See Section 4.1 for some background on VRF.) If the random number is below a threshold, it indicates that the participant has been elected as the leader, who then broadcasts the block along with the associated proof generated by the VRF to the network. Both Ouroboros Praos and Snow White require only one broadcast message to prove eligibility (as we do in Caucus in Chapter 4). Another comparable protocol is Algorand [38], which proposes a scalable Byzantine agreement protocol. In Algorand, a committee is elected at each step of the protocol by a "cryptographic sortition" that also uses VRF. Each committee member then gets the opportunity to vote on a block. The number of messages exchanged is $O(T \times \tau)$ where τ is the expected size

of the committee and T the expected fraction of honest players among them. The numerical values given for these parameters are $\tau = 2000$ and $T = 0.685$, assuming there are 80% of honest users in total. This protocol is thus less scalable than our protocol Fantômette that requires only one message broadcast. The cryptographic sortition used in Algorand inspired Caucus in Chapter 4.

As for the incentives, in Ouroboros and Ouroboros Praos [46], the honest strategy is a δ-Nash equilibrium. In a subsequent paper [48], they propose a reward-sharing scheme that ensures that the desired number of pools emerge, i.e., that the system is sufficiently decentralized. They do not provide any game-theoretic guarantees that consider irrational players.

In Snow White [47], the incentive structure is based on that of Fruitchains [49], where honest mining is a Nash Equilibrium (NE) resilient to coalitions. Similarly to Ouroboros they do not consider incentives with irrational players.

Algorand does not address the question of incentives.

Unlike Ouroboros Praos, Algorand and Snow-white, Fantômette does not require any synchronized clocks as liveness is achieved by the means of a Verifiable Delay Function [50] (i.e., the players wait for the VDF to compute in the case where no leaders are elected, they do not need to have synchronized clocks).

Casper [51] is still a work in progress, so it is difficult to say how well it addresses scalability. On the topic of incentivization, it proposes that following that protocol should be a Nash equilibrium and that an attacker should lose more in an attack than the victims of the attack. Our protocol uses a similar idea of adding a very harsh financial punishment to equivocating players.

Other non-academic work proposes PoS solutions [35, 52, 53, 54], which are related to Fantômette in terms of the recurrent theme of punishment in the case of misbehavior.

To summarise, unlike Fantômette, most existing proposals for non-PoW blockchain consensus protocols provide at most a basic game-theoretic analysis, using techniques like Nash equilibria, but do not consider more advanced game-theoretic properties that consider coalitions or the presence of Byzantine adver-

	Algorand	Snow White	Ouroboros Praos	Fantomette
# Messages	$T \times \tau$	1	1	1
Liveness	Synchronised Clocks	Synchronised Clocks	Synchronised Clocks	VDF
Equilibrium	NA	Coalition NE	Coalition NE	ε-robustness

Table 3.1: Comparison of different PoS consensus protocols

saries. Furthermore, Fantômette uses blockDAGs and an incentive scheme to move away from BFT-style algorithms, which are inherently less scalable. See Table 3.1 for a comparative summary of Algorand, Snow White, Ouroboros Praos and Fantômette. (See Chapter 4 for some background on robustness.)

3.3 BlockDAGs

SPECTRE [55] introduced the notion of a blockDAG, which was further refined by PHANTOM [56]. In a blockDAG, the block structure is a Directed Acyclic Graph (DAG) of blocks rather than a chain of blocks: a block can reference many other blocks, rather than just its parent as in Bitcoin. We provide a more in-depth background on BlockDAGs in Section 4.1.2. Our Fantômette protocol is inspired by PHANTOM (although translated to a non-PoW setting), and in particular we use their definition of *anticone* to classify blocks that are not connected well enough to the rest of the DAG. The motivation behind PHANTOM is to provide a system that has a better throughput than Bitcoin by having a faster block rate. The idea is that by using a DAG, it will be easy to detect "honest" blocks from adversarial blocks as honest blocks will form a well connected DAG as opposed to adversarial blocks which will have fewer connections, in the DAG, to the rest of the blocks. Informally, the protocol extracts a subDAG from the current DAG by removing nodes that are labeled adversarial, and then performs some topological ordering function on the remaining subDAG in order to obtain a full order on the blocks (rather than partial). Unlike Fantômette, PHANTOM does not have a natural ordering of blocks and thus requires a much more complex algorithm. In Fantômette, however, we use two different types of links in the blockDAG: bets and references. This allows us to maintain a notion of chain and thus use a rather simple fork choice rule.

Avalanche [57], a recent proposal also relying on blockDAG, but in the context of PoS, recently appeared. They propose a PBFT-style consensus protocol that achieves a communication complexity of $O(kn)$ for some $k \ll n$, and is thus more expensive than Fantômette that requires a single message broadcast. They do not consider the question of incentives.

As highlighted in this section and the previous one, most of the literature under-considers game-theoretic security notions when designing blockchain protocols. This leads to many failures on incentives that we highlight in the next section where we review the work that has been done on the incentives of blockchains.

3.4 Blockchain Security and Incentives

Nakamoto's original Bitcoin paper [2] provided only informal security arguments. Several papers have since formally argued the security of Bitcoin in different models [11, 58, 10] but without consideration of incentives in most cases.

In early work in this area, Kroll et al. [59] show that there is a NE in which all players behave consistently with Bitcoin's reference implementation, along with infinitely many equilibria in which they behave otherwise e.g., where they all agree to change a rule. Attacks like selfish mining [60, 12, 61] put this into question, showing that their model did not encompass behaviour that could realistically occur.

More recently Garay et al. [62] proved the security of Bitcoin in the Rational Protocol Design framework [63]. Their approach is based on the observation that Bitcoin is still working despite flaws, and they prove that Bitcoin is secure by relying on the rationality of players rather than an honest majority. This model still has some flaws, e.g., they do not consider fully malicious players. Moreover, their model does not encompass attacks that have been found on Bitcoin's incentive structure that we now describe.

Selfish mining [60] involves a rational miner increasing their expected utility by withholding their blocks instead of broadcasting them to the rest of the network, giving them an advantage in solving the new proof-of-work and making the rest of the network waste computation by mining on a block that is not the top of the chain.

Inspired by techniques introduced by Gervais et al. [64], Sapirshtein et al. [12] use Markov Decision Processes (MDP) to find the optimal strategy when doing selfish mining. (MDP are used to help make decisions in a discrete state space where outcomes are partially random.) They show that with this strategy, an adversary could mount a 51% attack with less than 25% of the computational power.

Another attack, the verifier dilemma [65] shows that miners are not incentivized to verify the content of blocks, especially when this incurs an important computation on their end.

Mining gaps are another type of attack on incentive structures [13, 66] where the time between the creation of blocks increases because miners wait to include enough transactions (in order to get the transaction fees). Both of these papers use simulations to quantify the attacks, using techniques such as no-regret learning where miners update their strategy at every "stage" of the game in a way to do as close to the best strategy as possible (had it been known from the beginning) or MDP.

Bribery attacks are another family of attacks, which are often thought of as an example of the *tragedy of the commons* that describes a situation when individuals acting selfishly affect the common good. In our context, it captures the fact that miners have to balance their aim to maximise their profit with the risk of affecting the long-term health of the cryptocurrency they mine, potentially reducing its price and their profit.

Bonneau [67] first proposed that an adversary could mount a 51% attack at a much reduced cost by renting the necessary hardware for the length of the attack rather than purchasing it permanently. Building on this idea, a briber could just pay existing miners to mine in a certain way, without ever needing to acquire any hardware. This leads to a series of papers [68, 69, 70, 71, 72] that show it is possible to introduce new incentives to an existing cryptocurrency, internally or externally, in ways that do not require trust between miners and bribers.

Mechanisms like Ethereum's uncle reward, which allows blocks that were mined but not appended to the blockchain to later be referenced in another block

for a reward can be used to subsidize the cost of bribery attacks [71] and selfish mining [73, 74]. This is unfortunate, as uncle rewards were originally introduced to aid decentralization [75] but have now been found to introduce incentives that work against this, by reducing the mining power required to perform certain attacks.

This puts into question the value of saying that a cryptocurrency is incentive compatible if new incentives can later be added. A cryptocurrency also does not exist in a vacuum, and external incentives can always manifest in adversarial ways. For example, Goldfinger attacks were proposed by Kroll et al. [59] and involve an adversary paying miners of a cryptocurrency to sabotage it by mining empty blocks. In some cases, even the threat of this type of attack can be enough to kill off a cryptocurrency, as users would not want their investment's value to deteriorate if the attack actually happens, and would thus not invest. As a Goldfinger attack can be implemented through a smart contract in another cryptocurrency [71], which gives payouts when an empty block is mined in the targeted cryptocurrency, it is not inconceivable that this could be attempted in practice. This clearly shows that incentives from outside the cryptocurrency itself must be considered.

Budish [76] proposes an economic analysis of 51% attacks and double spending, and shows that the Nakamoto consensus has inherent economic limitations. In particular, he shows from a strictly economic point of view that the security of the blockchain relies on scarce, non-repurposable resources (i.e., ASICs) used by miners as opposed to Nakamoto's vision of "one-CPU-one-vote", and that the blockchain is vulnerable to sabotage at a cost that is linear in the amount of specialized computational equipment devoted to its maintenance.

In order to solve the failures in the incentive structure just outlined, some new designs are being proposed. For example, blockDAGs that we already discussed previously. When creating a new block, a miner should point to all the blocks that they are aware of, revealing their view of the blockchain. This exposes more of the decision making of the players. Players could still however lie and pretend not to be aware of blocks but this is usually disincentivized by the protocol (i.e., a block that references more blocks will have a bigger reward). Players are thus incentivized to

reveal the truth. We leverage this concept in Fantômette.

Among other types of consensus protocols, there are a couple that do closely consider the topic of incentivization. SpaceMint [77] is a cryptocurrency based on proof-of-space, and they prove that following the protocol is a Nash equilibrium. The first version of Solidus [78], which is a consensus protocol based on PoW, provides a (k,t)-robust equilibrium for any $k+t < f$, although they leave a rigorous proof of this for future work. We will introduce the robustness solution concept in Chapter 4, where we use it for our security model.

3.5 Governance

From a theoretical point-of-view, De Filippi and Loveluck examine the overall decision-making process of the Bitcoin developers [79]. In particular, they discuss the "block size" debate and the difficulty in deciding whether or not to fork Bitcoin in order to increase the block size. Barrera and Hurder [80] propose a game-theoretic analysis of blockchain governance. They model the decision of implementing a new policy as a coordination game. They show that, under certain circumstances, hard forks and chain splits maximize the social welfare of a blockchain community and the problem of "tyranny of the majority" does not arise in the blockchain setting. Atzori gives a critical evaluation of whether blockchains are suitable as political tools [81], and examines to which extent they can mitigate coercion, centralization, and hierarchical structures. Reijers et al. study the question of governance from the perspective of social contract theory and finds that it fails to incorporate aspects of distributive justice [82]. Similarly, Lehdonvirta poses the "blockchain paradox," in which he argues that once you solve the problem of decentralized governance, you no longer need blockchains [83].

From a quantitative point-of-view, Srinivasan and Lee [84] introduce a metric for measuring the decentralization in cryptocurrencies which they call the Nakamoto coefficient. We go beyond this work in Chapter 5, and use their Nakamoto coefficient as well as other measurement techniques in order to compare and contrast the level of decentralization in Bitcoin and Ethereum.

3.6 Identity Management on Blockchains

Decentralized PKI In the setting of certificate issuance, the systems we propose in Chapter 6 are related to the idea of a public-key infrastructure (PKI). Some of our proposed registration protocols rely on a fixed set of specified registrars. These are related to the decentralized PKIs proposed by Fromknecht et al. [85] and the ARPKI system [86], which both distribute the process of certificate issuance to not only provide transparency into the process but also prevent misbehavior in the first place. We also provide more ad-hoc settings in which we allow any user to act as a registrar. This is related to the idea of the web of trust that we introduce later in Chapter 6.

Anonymous credentials The notion of accessing a service in a privacy-preserving manner can seemingly be achieved by anonymous credentials [87, 88, 89], which allow an issuer to create credentials that vouch for a user's identity or other generic attributes (e.g., their age). These credentials can then be shown to a verifier in a way that doesn't reveal anything to the verifier beyond the fact that the user possesses the attribute (e.g., is over 18 years old). The idea of issuing anonymous credentials has also been explored in the decentralized setting [90]. Our goal, however, is to allow users to not only access services but also to openly engage in existing blockchain-based systems using a registered identifier that — despite being vouched for by some registrar — cannot be linked to their real-world identity. To the best of our knowledge, this goal cannot be achieved directly by any solution based on anonymous credentials, at least not in an efficient manner: even if credentials could be issued on-chain, they would be larger than a blockchain address and issuance would consume a prohibitively high amount of gas.

Identity management on blockchains Finally, a lot of recent work, both in the academic literature and in the broader community, has focused on the question of using the blockchain to establish and manage identities (see, e.g., `https://github.com/peacekeeper/blockchain-identity` for a comprehensive list). The ChainAnchor project [91] presents a system for identity and access control, with the purpose of having anonymous but verified on-chain identities, and of providing in-

centives to miners to include only transactions from verified users. While some of the techniques used are similar to our own, as they also adopt a form of registration, their focus is on permissioned ledgers and on requiring registration for all users (which is useful in, e.g., the setting of providing compliance with know-your-customer and anti-money-laundering regulations). In terms of industrial solutions, uPort [92] is a web identity management system that links an Ethereum address with a name, profile picture, and other information like an email address or Twitter account, and OneName is a similar initiative that does the same with Bitcoin addresses. MIT also recently introduced its Digital Certificates Project [93] using the Bitcoin blockchain, with the goal of making "certificates transferable and more easily verifiable." These solutions have seen some level of adoption and we borrow some useful features from each of them (e.g., we use a similar technique to achieve revocation as the Digital Certificates project), but add the benefit of additional points of comparison, and a security framework and analysis. Posterior to our work, Hyperledger Indy [94] is a project for interoperability of decentralized identities on blockchains. Coconut proposes anonymous credentials on blockchains [95], using Non-Interactive Zero Knowledge Proofs.

Chapter 4

Betting on Blockchain Consensus with Fantômette

In this chapter, we look at the main component of a decentralized cryptocurrency: its consensus protocol. We present a consensus protocol, Fantômette, based on proof-of-stake that does not require any hardware investment from participants and thus lowers the barrier to entry, compared to proof-of-work. Additionally, this protocol consumes far less electricity than proof-of-work, so it will be cheaper to run than proof-of-work and is less likely to lead to pools as a decrease of variance will have less impact on participants.

At heart, one of the biggest obstacles in designing proof-of-stake protocols is in scaling their underlying *leader election* protocol, in which one participant or subset of participants is chosen to lead the decisions around what information should get added to the ledger for a single round (or set period of time). The second novel requirement of blockchains is the explicit economic incentivization on behalf of participants. More specifically, the protocol should not incentivize the formation of pools and should be the best strategy to follow for rational players.

Our contributions

We propose Fantômette, a new proof-of-stake consensus protocol that incorporates an incentive design and works in a setting that considers both rational and Byzantine adversaries. Our initial observation is that the PoW-based setting contains an implicit investment on the part of the miners, in the form of the costs of hard-

ware and electricity. In moving away from PoW, this implicit investment no longer exists, giving rise to new potential attacks due to the fact that creating blocks is now costless. It is thus necessary to compensate by adding explicit punishments into the protocol for participants who misbehave. This is difficult to do in a regular blockchain setting. In particular, blockchains do not reveal information about which other blocks miners may have been aware of at the time they produced their block, so they cannot be punished for making "wrong" choices. We thus move to the setting of *blockDAGs* [55, 56], which induce a more complex fork-choice rule and expose more of the decision-making process of participants. Within this model, we are able to leverage the requirement that players must place *security deposits* in advance of participating in the consensus protocol to achieve two things. First, we can implement punishments by taking away some of the security deposit, and thus incentivize rational players to follow the protocol. Second, because this allows the players to be identified, we can provide a decentralized *checkpointing* system, which in turn allows us to achieve a notion of finality.

Along the way, we present in Section 4.3 a leader election protocol, Caucus, that is specifically adapted for open blockchains, and that we prove secure in a model presented in Section 4.2. We then use Caucus as a component in the broader Fantômette consensus protocol, which we present in Section 4.4 and argue for the security of, according to our specific threat model, in Section 4.5. Here we rely on Caucus to address the first requirement of scaling in blockchain-based consensus protocols, so we can focus almost entirely on the second requirement of incentivization. Our protocol includes a decentralized checkpointing mechanism that is inferred from the structure of the blockDAGs and could be of independent interest in the context of blockDAGs. This chapter is based on [20].

4.1 Background

In this section, we give some additional background on concepts used in the rest of this chapter.

4.1.1 Additional Cryptographic Primitives

4.1.1.1 Coin tossing and random beacons

Coin tossing allows two or more parties to agree on a single or many random bits [96, 97]; i.e., to output a value R that is computationally indistinguishable from random.

A coin-tossing protocol must satisfy *liveness*, *unpredictability*, and *unbiasability* [98], where we define these as follows:

Definition 4.1.1. *Let f_a be the fraction of participants controlled by an adversary $\mathcal{A}$. Then a coin-tossing protocol satisfies* f_a-liveness *if it is still possible to agree on a random value R even in the face of such an $\mathcal{A}$.*

Definition 4.1.2. *A coin-tossing protocol satisfies* unpredictability *if, prior to some step* barrier *in the protocol, no PT adversary can produce better than a random guess at the value of R.*

Definition 4.1.3. *A coin-tossing protocol is* f_a-unbiasable *if for all PT adversaries $\mathcal{A}$ controlling an f_a fraction of participants, the output R is still computationally indistinguishable from a uniformly distributed random string.*

Many coin-tossing protocols [96] follow a structure called *commit-then-reveal*: to start, in the commit phase each participant creates and broadcasts a cryptographic commitment to a random value. In the reveal phase, participants broadcast the opening of their commitments, which can be used to check the validity of the initial commitment. The output value is then some combination (e.g., XOR) of all the individual random values. Intuitively, while this basic solution does not satisfy liveness, it satisfies unpredictability due to the hiding property of the commitment scheme (where barrier corresponds to the step in which the last commitment is opened), and unbiasability due to the binding property. In order to achieve liveness, *secret sharing* can be used. To ensure that the protocol produces a valid output, participants create shares of their secret and send these shares to other participants during the commit phase. This allows participants to recover the value of another participant even if they abort.

A closely related concept to coin tossing is *random beacons*. These were first introduced by Rabin [99] as a service for "emitting at regularly spaced time intervals, randomly chosen integers". To extend the above definitions to random beacons, as inspired by [100], we require that the properties of f_a-liveness and f_a-unbiasability apply for each iteration of the beacon, or *round*. We also require that the barrier in the unpredictability definition is at least the beginning of each round.

4.1.1.2 Verifiable Random Function (VRF)

Another concept related to coin tossing is *Verifiable Random Function*. Verifiable Random Functions (VRF), first introduced by Micali et al. [101], generate a pseudo-random number in a publicly verifiable way. Formally, a VRF is defined as follows:

Definition 4.1.4. *A VRF consists of three polynomial-time algorithm* $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ *that works as follows: (1)* $\mathsf{Gen}(1^\lambda)$ *outputs a key pair* (pk, sk)*; (2)* $\mathsf{Prove}_{sk}(x)$ *outputs a pair* $(y = G_{sk}(x), p = p_{sk}(x))$*; (3)* $\mathsf{Verify}(x, y, p)$ *outputs 0 or 1 and verifies that* $y = G_{sk}(x)$ *using p. A VRF satisfies correctness if:*

- *if* $(y, p) = \mathsf{Prove}_{sk}(x)$ *then* $\mathsf{Verify}(x, y, p) = 1$
- *for every* (sk, x) *there is a unique y such that* $\mathsf{Verify}(x, y, p_{sk}(x)) = 1$
- *it verifies pseudo-randomness: for any PPT algorithm* $A = (A_E, A_J)$*:*

$$\mathbb{P}\left[\begin{array}{l|c} b = b' & \begin{array}{c}(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda);\\ (x, A_{st}) \leftarrow A_E^{\mathsf{Prove}(.)}(pk);\\ y_0 = G_{sk}(x); y_1 \leftarrow \{0,1\}^{len(G)};\\ b \leftarrow \{0,1\}; b' \leftarrow A_J^{\mathsf{Prove}(.)}(y_b, A_{st})\end{array}\end{array}\right] \leq \frac{1}{2} + negl(k)$$

We ask, furthermore, that the above properties hold even in the case where an adversary has maliciously generated their keypairs. Such schemes are presented in [38, 46].

4.1.1.3 Verifiable Delay Function (VDF)

Informally, a Verifiable Delay Function requires a sequential number of steps to compute such that it is not feasible for an adversary to predict the output of the

function faster than the specified amount of time, even if they possess a large number of parallel processors. The simplest example of such a function is $F = H^p$; i.e., a hash function iterated p times. The drawback of such a function, however is that it would also take the same amount of time for someone to verify the correctness of the function. Bünz et al. proposed an efficient Verifiable Delay function [50] that can be efficiently and publicly verified yet requires sequential steps to compute.

More formally, a VDF is defined by three algorithms (VDF.Setup, Eval, VDF.Verify) such that (1) VDF.Setup returns an evaluation and verification key (ek, vk); (2) Eval takes an input x and produces an output y and proof VDF.π; (3) VDF.Verify takes input x, output $y = F(x)$ and proof VDF.π and return 1 if y and VDF.π were generated correctly and 0 otherwise. A VDF must satisfy three properties [102], (1) evaluation time: for every x, Eval runs in time at most δ; (2) sequentiality: a parallel algorithm that runs in time less than δ cannot compute the function; (3) uniqueness: for an input x, exactly one y will be accepted by VDF.Verify.

VDFs are still a work in progress, with new constructions being proposed. In this work, we use them only as a fallback in the case where no leader is elected. There could be other ways to update the random beacon when no leader is elected than using a VDF but the advantage of a VDF is that it forces all the players to wait for the same amount of time (without requiring synchronized clocks). It will thus ensure that any elected player will have enough time to reveal themselves before the rest of the players move on. The VDF will, however, not appear in most of our security arguments as we consider a limited adversary that we will specify in section 4.5.1. The only place where the VDF will appear in our security arguments is when we argue for liveness (i.e., no adversary can force the protocol to completely stop).

4.1.2 BlockDAGs

As the name suggests, a blockchain is a chain of blocks, with each block referring to only one previous block. In contrast, a *blockDAG* [55, 56], is a directed acyclic graph (DAG) of blocks. In this thesis (which slightly adapts the original definitions),

every block still specifies a single parent block, but can also *reference* other recent blocks of which it is aware. These are referred to as *leaf blocks*, as they are leaves in the tree (also sometimes referred to as the tips of the chain), and are denoted $\mathsf{Leaves}(\mathsf{G})$. Blocks thus have the form $\mathsf{B} = (\mathsf{B}_{\mathsf{prev}}, \mathcal{B}_{\mathsf{leaf}}, \pi, \mathsf{txset})$ where $\mathsf{B}_{\mathsf{prev}}$ is the parent block, $\mathcal{B}_{\mathsf{leaf}}$ is a list of previous leaf blocks, π is a proof of some type of eligibility (e.g., a PoW), and txset is the transactions contained within the block. We denote by $\mathsf{B}.\mathsf{snd}$ the participant that created block B. In addition, we define the following notation:

- G denotes a DAG, G^i the DAG according to a participant i, and $\mathcal{G}$ the space of all possible DAGs.
- In a block, $\mathsf{B}_{\mathsf{prev}}$ is a direct *parent* of B, and $\mathsf{Ancestors}(\mathsf{B})$ denotes the set of all blocks that are parents of B, either directly or indirectly. By convention we assume $\mathsf{B} \in \mathsf{Ancestors}(\mathsf{B})$.
- A *chain* is a set of blocks $\mathcal{M}$ such that there exists one block B for which $\mathcal{M} = \mathsf{Ancestors}(\mathsf{B})$.
- $\mathsf{Past}(\mathsf{B})$ denotes the subDAG consisting of all the blocks that B references directly or indirectly. By convention we assume $\mathsf{B} \in \mathsf{Past}(\mathsf{B})$.
- $\mathsf{DirectFuture}(\mathsf{B})$ denotes the set of blocks that directly reference B (i.e., that include it in $\mathcal{B}_{\mathsf{leaf}}$).
- $\mathsf{Anticone}(\mathsf{B})$ denotes the set of blocks B' such that $\mathsf{B}' \notin \mathsf{Past}(\mathsf{B})$ and $\mathsf{B} \notin \mathsf{Past}(\mathsf{B}')$.
- d denotes the *distance* between two blocks in the DAG.
- The *biggest common prefix DAG* (BCPD) is the biggest subDAG that is agreed upon by more than 50% of the players.

For example, if we consider the blockDAG in Figure 4.1, we have that $\mathsf{Ancestors}(H) = \{E, A, \text{genesis}\}$ and $\{H, E, A, \text{genesis}\}$ forms a chain. We also have that $\mathsf{DirectFuture}(F) = \{H, I\}$, $\mathsf{Past}(H) = \{E, F, A, B, C, \text{genesis}\}$, $\mathsf{Anticone}(E) = \{D, F, G, I\}$, and $d(A, H) = 2$.

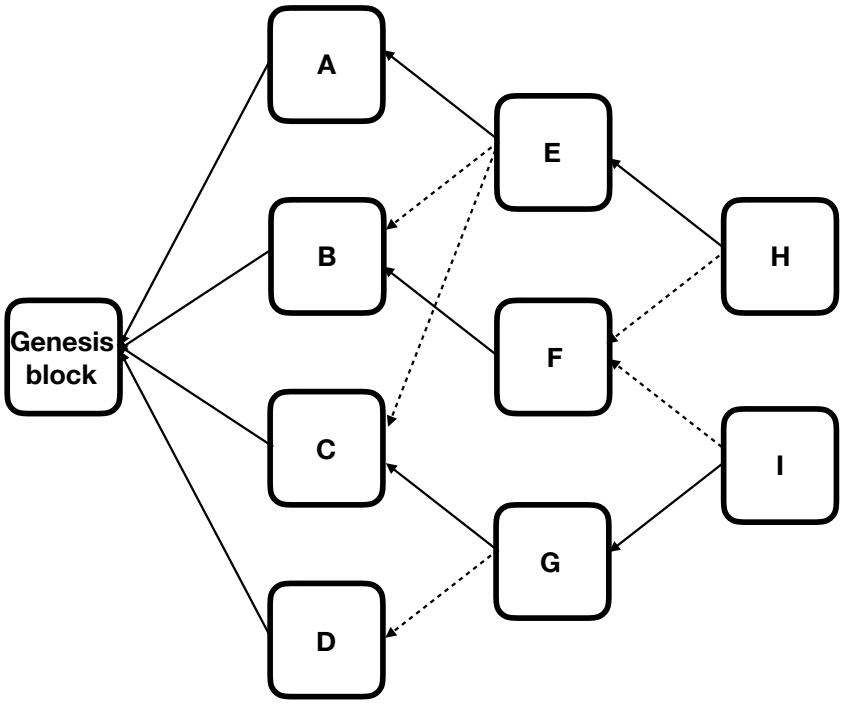


Figure 4.1: Example of a blockDAG. A full arrow indicates a bet ($\mathsf{B}_{\mathsf{prev}}$) and a dashed arrow indicates a reference ($\mathcal{B}_{\mathsf{leaf}}$).

4.1.3 Consensus as a game

In game-theoretic terms, we consider the consensus protocol as a game in infinite extensive-form with imperfect information [103], where the utilities of the player depend on the blocks they have contributed. A blockchain-consensus game theoretically has an infinite horizon, but here we consider a finite horizon of some unknown length T. We assume that at each node in the game tree player i is in some local state that includes their view of the blockDAG and some private information. We note $\mathcal{P}$ the set of participants.

Following Abraham et al. [104], we consider the following framework. With each run of the game that results in a blockDAG G, we associate some *utility* with player i, denoted $u_i(\mathsf{G})$. A *strategy* σ_i for player $i \in \mathcal{P}$ is a (possibly randomized) function from i's local state to some set of actions, and tells the player what to do at each step. A *joint* strategy $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ for the players determines a distribution over paths in the game tree. Let $u_i(\vec{\sigma})$ denote player i's expected utility if $\vec{\sigma}$ is played. Let $\mathcal{S}_i$ denotes the set of possibles strategies for player i, and let $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$. A joint strategy is a *Nash equilibrium* if no player can gain any

advantage by using a different strategy, given that all the other players do not change their strategies.

An extension of Nash equilibria is *coalition-proof* Nash equilibria [104]. A strategy is *k-resilient* if a coalition of up to k players cannot increase their utility function by deviating from the strategy, given that other players follow the strategy.

Definition 4.1.5 (k-resilient strategy)**.** *Given a non-empty set of players $\mathcal{C} \subseteq \mathcal{P}, \vec{\sigma}_{\mathcal{C}} \in \mathcal{S}_{\mathcal{C}}$ is a group best response for $\mathcal{C}$ to $\vec{\sigma}_{-\mathcal{C}} \in \mathcal{S}_{-\mathcal{C}}$ if for all $\vec{\tau}_{\mathcal{C}} \in \sigma_{\mathcal{C}}$ and $i \in \mathcal{C}$, we have:*

$$u_i(\vec{\sigma}_{\mathcal{C}}, \vec{\sigma}_{-\mathcal{C}}) \geq u_i(\vec{\tau}_{\mathcal{C}}, \vec{\sigma}_{-\mathcal{C}})$$

A joint strategy is a k-resilient strategy if for all $\mathcal{C} \subseteq \mathcal{P}$ with $|\mathcal{C}| \leq k$, $\vec{\sigma}_{\mathcal{C}}$ is a group best response for C to $\vec{\sigma}_{-\mathcal{C}}$.

A strategy is *t-immune* if, even when a group of size t of players deviates arbitrarily from the protocol, the payoff of the non-deviating players is greater than or equal to their payoff in the case where these players do not deviate.

Definition 4.1.6 (t-immune strategy)**.** *A joint strategy $\vec{\sigma} \in \mathcal{S}$ is t-immune if, for all $T \subseteq \mathcal{P}$ with $|T| \leq t$, all $\vec{\tau}_T \in \mathcal{S}_T$, and all $i \notin T$ we have $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T) \geq u_i(\vec{\sigma})$.*

A strategy is a (k,t)-*robust equilibrium* if it is a k-resilient and t-immune strategy. Similarly, in an ε-(k,t)-robust equilibrium, is an equilibrium that is ε-k-resilient and ε-t-immune according to the definitions below.

Definition 4.1.7 (ε-k-resilient strategy)**.** *Given a non-empty set $\mathcal{C} \subseteq \mathcal{P}, \vec{\sigma}_{\mathcal{C}} \in \mathcal{S}_{\mathcal{C}}$ is an ε-best response for $\mathcal{C}$ to $\vec{\sigma}_{-\mathcal{C}} \in \mathcal{S}_{-\mathcal{C}}$ if for all $\vec{\tau}_{\mathcal{C}} \in \sigma_{\mathcal{C}}$ and $i \in \mathcal{C}$, we have:*

$$u_i(\vec{\sigma}_{\mathcal{C}}, \vec{\sigma}_{-\mathcal{C}}) \geq \varepsilon u_i(\vec{\tau}_{\mathcal{C}}, \vec{\sigma}_{-\mathcal{C}})$$

A joint strategy is an ε-$k-$resilient strategy if for all $\mathcal{C} \subseteq \mathcal{P}$ with $|\mathcal{C}| \leq k$, $\vec{\sigma}_{\mathcal{C}}$ is an ε-best response for C to $\vec{\sigma}_{-\mathcal{C}}$.

Definition 4.1.8 (ε-t-immune strategy)**.** *A joint strategy $\vec{\sigma} \in \mathcal{S}$ is ε-t-immune if, for all $T \subseteq \mathcal{P}$ with $|T| \leq t$, all $\vec{\tau}_T \in \mathcal{S}_T$, and all $i \notin T$ we have $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T) \geq \varepsilon u_i(\vec{\sigma})$.*

In addition to the players, we assume there exist some other agents that do not participate in the game but still maintain a view of the blockDAG. These *passive* agents represent full nodes, and will not accept blocks that are clearly invalid. The existence of these agents allows us to assume that even adversarial players must create valid blocks.

4.2 Modelling Blockchain Consensus

We present in this section a model for blockchain-based consensus, run amongst a set of participants (also called players) $\mathcal{P}$. A block B is considered to be a bet on its ancestors $\mathsf{Ancestors}(\mathsf{B})$. After introducing the assumptions we make about participants, we present a model for leader election, which is used to determine which participants are eligible to create a block. We then present a model for blockchain consensus, along with its associated security notions.

4.2.1 Assumptions

Following the related literature [46, 47], we consider a semi-synchronous model [105]: time is divided in units called slots, and each message is delivered within a maximum delay of Δ slots (for Δ unknown to participants). We assume that all players form a well-connected network, in a way similar to Bitcoin, where they can broadcast a message to their peers, and that communication comes "for free."

Types of players We follow the BAR model [106], which means players are either (1) Byzantine, meaning that they behave in an arbitrary way; (2) altruistic, meaning that they follow the protocol; or (3) rational, meaning that they act to maximize their expected utility. Although in our security properties we do not consider all three types of players at the same time, we explain at the end of this section why this is enough to argue that our protocol works in the BAR model. We use (f_B, f_A, f_R) to denote the respective fractions of Byzantine, altruistic and rational players. Previous research has shown that altruistic behavior is indeed observed in real-world systems (like Tor or torrenting) [107], so is reasonable to consider. In addition to these types, as stated in Section 4.1.3 we consider *passive* participants.

They represent the users of a currency who keep a copy of the blockchain and passively verify every block (i.e., the full nodes). They will not explicitly appear in the protocol, rather we assume that they "force" the creation of valid blocks as we explain in Section 4.5, since if the chain includes invalid blocks or other obvious forms of misbehavior they will simply abandon or fork the currency. Unless specified otherwise, players will now mean active players.

We consider a semi-permissionless setting, meaning that everyone is allowed to join the protocol but they have to place a *security deposit* locking some of their funds to do so; if they wish to leave the protocol, then they must wait for some period of time before they can access these funds again. This allows us to keep the openness of decentralization while preventing Sybils. Moreover, for ease of exposition, we consider a flat-model meaning that one participant accounts for one *unit* of stake. Thus saying two-third of participants is equivalent to saying participants that together own two-third of the stake that is in deposit. Our protocol can however be extended easily to a non-flat model, we discuss how at the end of Section 4.3.1. Most of the paper makes the assumptions of a dynamic committee where participants can leave and join as explained above. However, to consider a notion of finality, we will need to strengthen this assumption. We will detail these assumptions in Section 4.4, but briefly we will allow for a reconfiguration period, and assume that outside of this period the set of participants is fixed.

4.2.2 A model for leader election

Most of the consensus protocols in the distributed systems literature are leader-based, as it is the optimal solution in terms of coordination [108]. Perhaps as a result, leader election has in general been very well studied within the distributed systems community [109, 110, 111, 112, 113]. Nevertheless, to the best of our knowledge the problem of leader election has not been given an extensive security-focused treatment, so in this section we provide a threat model in which we consider a variety of adversarial behavior.

Each participant maintains some private state $\mathsf{st}_{\mathsf{priv}}$, and their view of the public state $\mathsf{st}_{\mathsf{pub}}$. For ease of exposition, we assume each $\mathsf{st}_{\mathsf{priv}}$ includes the public state

$\mathsf{st}_{\mathsf{pub}}$. We refer to a message sent by a participant as a *transaction*, denoted tx, where this transaction can either be broadcast to other participants (as in a more classical consensus protocol) or committed to a public blockchain.

Our model consists of three algorithms and one interactive protocol, which behave as follows:

$(\mathbf{st}_{\mathsf{priv}}, \mathsf{tx}_{\mathsf{com}}) \xleftarrow{\$} \mathsf{Commit}(\mathbf{st}_{\mathsf{pub}})$ is used by a participant to commit themselves to participating in the leader election. This involves establishing both an initial private state $\mathsf{st}_{\mathsf{priv}}$ and a public announcement $\mathsf{tx}_{\mathsf{com}}$.

$\{\mathbf{st}^{(i)}_{\mathsf{priv}}\}_i \xleftarrow{\$} \mathtt{Update}(1^\lambda, \mathcal{P}, \{(\mathsf{rnd}, \mathbf{st}^{(i)}_{\mathsf{priv}})\}_i)$ is run amongst the committed participants, each of whom is given rnd and their own private state $\mathsf{st}^{(i)}_{\mathsf{priv}}$, in order to update both the public state $\mathsf{st}_{\mathsf{pub}}$ and their own private states to prepare the leader election for round rnd.

$\mathsf{tx}_{\mathsf{rev}} \xleftarrow{\$} \mathsf{Reveal}(\mathsf{rnd}, \mathbf{st}_{\mathsf{priv}})$ is used by a participant to broadcast a proof of their eligibility $\mathsf{tx}_{\mathsf{rev}}$ for round rnd (or $\perp$ if they are not eligible).

$0/1 \leftarrow \mathsf{Verify}(\mathbf{st}_{\mathsf{pub}}, \mathsf{tx}_{\mathsf{rev}})$ is used by a participant to verify a claim $\mathsf{tx}_{\mathsf{rev}}$.

We would like a leader election protocol to achieve three security properties: *liveness*, *unpredictability*, and *fairness*. The first property maps to the established property of liveness for classical consensus protocols, although as we see below we consider several different flavors of unpredictability that are specific to the blockchain-based setting. The final one, fairness (related to *chain quality* [10]), is especially important in open protocols like blockchains, in which participation must be explicitly incentivized rather than assumed.

We begin by defining liveness, which requires that a leader can be elected even if some fraction of participants are malicious or inactive.

Definition 4.2.1 (Liveness)**.** *Let f_a be the fraction of participants controlled by an adversary $\mathcal{A}$. Then a leader election protocol satisfies* f_a-liveness *if it is still possible to elect a leader even in the face of such an $\mathcal{A}$; i.e., if for every public state st_{pub} that has been produced via* `Update` *with the possible participation of $\mathcal{A}$, it is*

still possible for at least one participant, in a round rnd*, to output a value* $\mathsf{tx}_{\mathsf{rev}}$ *such that* $\mathsf{Verify}(st_{\mathsf{pub}}, \mathsf{tx}_{\mathsf{rev}}) = 1$.

Unpredictability requires that participants cannot predict which participants will be elected leaders before some time.

Definition 4.2.2 (Unpredictability)**.** *A leader election protocol satisfies* unpredictability *if, prior to some step* barrier *in the protocol, no PT adversary* $\mathcal{A}$ *can produce better than a random guess at whether or not a given participant will be eligible for round* rnd*, except with negligible probability. If* barrier *is the step in which a participant broadcasts* $\mathsf{tx}_{\mathsf{rev}}$*, and we require* $\mathcal{A}$ *to guess only about the eligibility of honest participants (rather than participants they control), then we say it satisfies* delayed *unpredictability. If it is still difficult for* $\mathcal{A}$ *to guess even about their own eligibility, we say it satisfies* private *unpredictability.*

Most consensus protocols satisfy only the regular variant of unpredictability we define, where barrier is the point at which the `Update` interaction is "ready" for round rnd (e.g., the participants have completed a coin-tossing).

If an adversary is aware of the eligibility of other participants ahead of time, then it may be able to target these specific participants for a denial-of-service (DoS) attack, which makes achieving liveness more difficult. A protocol that satisfies delayed unpredictability solves this issue, however, as participants reveal their own eligibility only when they choose to do so, by which point it may be too late for the adversary to do anything. E.g., in a proof-of-stake protocol, if participants include proofs of eligibility only in the blocks they propose, then by the time the leader is known the adversary has nothing to gain by targeting them for a DoS attack. Similarly, an adversary cannot know which participants to corrupt in advance because it does not know if they will be eligible. This helps to obtain security against a fully adaptive adversary that is able to dynamically update the set of participants it is corrupting.

A protocol that satisfies private unpredictability, in contrast, is able to prevent an adversary from inflating their own role as a leader. For example, if an adversary can predict many rounds into the future what their own eligibility will be, they

may attempt to bias the protocol in their favor by grinding through the problem space in order to produce an initial commitment $\mathsf{tx}_{\mathsf{com}}$ that yields good future results. Additionally, private unpredictability ensures that participants stay online for the duration of the protocol as they may miss their turns otherwise. This could be helpful, for example, for detecting and slashing malicious behaviour or for specific use-cases, such as Filecoin[1] that is based on Proof-of-Space and where an adversary could cheat the protocol by faking their storage if they are aware of their own eligibility ahead of time.

Lastly, fairness requires that each honest committed participant is selected as leader equally often. While for the sake of simplicity our definition considers equal weighting of participants, it can easily be extended to consider participants with respect to some other distribution (e.g., in a proof-of-stake application, participants may be selected as leader in proportion to their represented "stake" in the system).

Definition 4.2.3 (Fairness)**.** *A leader election protocol is fair if for all PT adversaries $\mathcal{A}$ the probability that one honest leader is selected as leader is nearly uniform; i.e., for all* rnd*,* st_{priv}*,* st_{pub} *(where again* st_{pub} *has been produced by* `Update` *with the possible participation of $\mathcal{A}$), and* $\mathsf{tx}_{\mathsf{rev}}$ *produced by an honest party,* $Pr[\mathsf{Verify}(st_{\mathsf{pub}}, \mathsf{tx}_{\mathsf{rev}}) = 1] \approx 1/n$.

4.2.3 Blockchain-based consensus

As with the leader election, each participant maintains some private state $\mathsf{st}_{\mathsf{priv}}$ and some view of the public state $\mathsf{st}_{\mathsf{pub}}$. We consider the following set of algorithms run by participants:

$\mathbf{st}_{\mathsf{priv}} \xleftarrow{\$} \mathsf{Setup}(1^\lambda)$ is used to establish the initial state: a public view of the blockchain (that is the same for every player) and their private state. This includes the Commit phase of the leader election protocol.

$\pi \xleftarrow{\$} \mathsf{Eligible}(\mathsf{B}, \mathbf{st}_{\mathsf{priv}})$ is run by each participant to determine if they are eligible to place a bet on a block B. If so, the algorithm outputs a proof π (and if not it outputs

[1] `https://filecoin.io/`

$\perp$), that other participants can verify using $\mathsf{Verify}(\mathrm{st}_{\mathsf{pub}}, \pi)$. The Verify algorithm is the same as the one used in the leader election protocol, where π=$\mathsf{tx}_{\mathsf{rev}}$.

$\mathsf{B} \xleftarrow{\$} \mathsf{Bet}(\mathbf{st}_{\mathsf{priv}})$ is used to create a new block and bet on some previous block.

$\mathsf{B} \leftarrow \mathsf{FCR}(\mathsf{G})$ defines the fork-choice rule that dictates which block altruistic players should bet on. To do so, it gives an explicit *score* to different chains, and chooses the tip of the chain with the biggest score.

$0/1 \leftarrow \mathsf{VerifyBlock}(\mathsf{G}, \mathsf{B})$ determines whether or not a block is valid, according to the current state of the blockDAG.

$\mathsf{M} \leftarrow \mathsf{Label}(\mathsf{G})$ defines a function $\mathsf{Label} : \mathcal{G} \to \mathcal{M}$ that takes a view of the blockDAG and associates with every block a label in $\{\text{winner}, \text{loser}, \text{neutral}\}$. This is crucial for incentivization, and is used to determine the reward that will be associated with every block. With each map Label, and player i, we associate a utility function u_i^{Label} that takes as input a blockDAG and outputs the utility of player i for that blockDAG. We will write $u_i^{\mathsf{Label}} = u_i$ if the label function is clear from context. The list of winning blocks constitutes a chain, which we call the *main chain*. Every chain of blocks that is not the main chain is called a *fork*.

We would like a blockchain consensus protocol to satisfy a few security properties. Unlike with the leader election, these have been carefully considered in the cryptographic literature [10, 11]. Pass et al. defined four desirable properties of blockchain consensus protocols [11]: consistency, future self-consistency, chain growth, and chain quality.

Chain growth [10] corresponds to the concept of liveness in distributed systems, and says the chain maintained by honest players grows with time. Consistency (also called common prefix [10]) and future self-consistency both capture the notion of safety traditionally used in the distributed systems literature. Consistency states that any two honest players should agree on their view of the chain except for the last Y blocks, and future self-consistency states that a player and their "future self" agree on their view of the chain except for the last Y blocks (the idea being that a player's chain will not change drastically over time). Here, we consider not

just a blockchain-based consensus protocol, but in fact one based on a blockDAG. Participants thus do not keep only the longest chain, but all the blocks they receive, which makes it difficult to use these definitions as is. Instead, we use the notion of *convergence*, which states that after some time, player converges towards a chain, meaning that no two altruistic players diverge on their view of the main chain except perhaps for the last blocks, and that a block that is in the main chain at some time τ_0 will still be in the main chain for any time $t > \tau_0$. We also ask that this condition holds for a chain of any length, thus capturing the chain growth (or liveness) in the same definition. More formally we define convergence as follows:

Definition 4.2.4 (Convergence)**.** *For every $k_0 \in \mathbb{N}$, there exists a chain $\mathsf{C}^0_{k_0}$ of length k_0 and time τ_0 such that: for all altruistic players i and time $t > \tau_0$: $\mathsf{C}^0_{k_0} \subseteq \mathsf{C}^{i,t}$, except with negligible probability, where $\mathsf{C}^{i,t}$ denotes the main chain of i at time t.*

Chain quality [10] corresponds to the notion of fairness, and says that honest players contribute some meaningful fraction of all blocks in the chain.

Definition 4.2.5 (Chain quality)**.** *Chain quality, parameterized by μ, says that an adversary controlling a fraction f_B of Byzantine players can contribute at most a fraction $1-\mu$ of the blocks in the main chain, except with small probability.*

Finally, we define a relatively unexplored property in blockchain consensus, *ε-robustness*, that explicitly captures the incentivization mechanism. The security notion is not considered by any consensus protocols (except [78] that states that they leave the proof for future work) and is paramount to capture the security of systems where incentives are at the core.

Definition 4.2.6 (ε-robustness)**.** *A protocol is ε-robust if given some fractions (f_B, f_A, f_R) of BAR players, following the protocol is a ε-(f_R, f_B)-robust equilibrium.*

A couple of remarks are in order: convergence and chain quality consider a Byzantine adversary. These are the more traditional security definitions that do not consider incentives or rationality.

As for robustness, the resiliency property considers a coalition of f_R rational players (while the rest of the players are altruistic). The immunity property, on the other hand, considers a coalition of f_B of Byzantine players, while the rest of the players are altruistic. It is hence worth noting that the treatment of rational and Byzantine adversaries is decoupled and they are not considered at the same time in any of our security definitions. Only the resiliency property considers explicit rational players while the other properties consider Byzantine vs altruistic players. Treating Byzantine and rational players simultaneously is an open problem however we note that having a protocol that is $(k,t)-$robust allows us to state that following the protocol is the rational thing to do and hence that rational players behave honestly (and are indistinguishable from honest users). This is why we state that our protocol works in the BAR setting, even though the different types of players are never considered simultaneously in any of our properties.

4.3 Caucus: A Leader Election Protocol

In this section, we present Caucus, a leader election protocol with minimal coordination that satisfies fairness, liveness, and strong notions of unpredictability. We note that this part of the protocol is modular and that any leader election that verifies the same security properties could work. Let's note that like other leader elections in PoS protocols [46, 38], an adversary controlling many winning candidates could choose which one they reveal and thus influence the leader election, at least among the players they control. We discuss this problem at the end of the section. We also note that our security properties do not require uniqueness of the leader election, i.e., there could be more than one leader elected at each round.

4.3.1 Our construction

The full Caucus protocol is summarized in Figure 4.2. We use a leader election that is similar to the one used in Algorand [38] (cryptographic sortition). We, however, use Verifiable Delay Functions (VDF) to achieve liveness. VDF also allows us to avoid synchronized clocks, like Algorand does. They ensure that every player waits the same time when no leader is elected, even if they don't have synchronized

Commit**:** A participant commits to their VRF secret key sk by creating a Commit transaction $\mathsf{tx}_{\mathsf{com}}$ that contains the VRF public key pk. Each broadcast commitment is added to a list $\mathtt{c}$ maintained in $\mathsf{st}_{\mathsf{pub}}$, and that participant is considered eligible to be elected leader after some fixed number of rounds have passed.

`Update`**:** Once enough participants are committed, participants run a secure coin-tossing protocol to obtain a random value R_1. They output a new $\mathsf{st}_{\mathsf{pub}} = (\mathtt{c}, R_1)$. **This interactive protocol is run only for** $\mathsf{rnd} = 1$**.**

Reveal**:** For $\mathsf{rnd} > 1$, every participant verifies their own eligibility by checking if $H(y_{\mathsf{rnd}}) < \mathsf{target}$, where $y_{\mathsf{rnd}} = G_{sk}(R_{\mathsf{rnd}})$ and $\mathsf{target} = H_{\mathsf{max}}/n_{\mathsf{rnd}}$. (Here n_{rnd} is the number of eligible participants; i.e., the number of participants that have committed a sufficient number of rounds before rnd and possibly have not been elected leader in the previous rounds.) The eligible participant, if one exists, then creates a transaction $\mathsf{tx}_{\mathsf{rev}}$ with their data y_{rnd} and proof $p_{\mathsf{rnd}} = p_{sk}(R_{\mathsf{rnd}})$ and broadcasts it to their peers.

Verify**:** Upon receiving a transaction $\mathsf{tx}_{\mathsf{rev}}$ from a participant i, participants extract y_{rnd} and p_{rnd} from $\mathsf{tx}_{\mathsf{rev}}$ and check whether or not $\mathsf{Verify}_{\mathrm{VRF}}(R_{\mathsf{rnd}}, y_{\mathsf{rnd}}, p_{\mathsf{rnd}}) = 1$. If these checks pass, then the public randomness is updated as $R_{\mathsf{rnd}+1} \leftarrow y_{\mathsf{rnd}}$ and they output 1, and otherwise the public state stays the same and they output 0.

Figure 4.2: The Caucus protocol.

clocks.

We assume participants have generated signing keypairs and are aware of the public key associated with each other participant (which can easily be achieved at the time participants run Commit). We omit the process of generating keys from our formal descriptions.

To be considered as potential leaders, participants must place a *security deposit*, which involves creating a commitment to a Verifiable Random Function's

(VRF) secret key. We choose a VRF that maintains its pseudo-randomness property even when an adversary maliciously computes their key pairs as has been proposed in [38, 46]. The Commit function runs $\mathsf{Gen}_{\mathrm{VRF}}$ and returns the VRF secret key as the private state of the participant and the VRF public key (incorporated into a transaction) as the transaction to add to a list of commitments c in the public state.

In the first round, participants must interact to establish a shared random value. This can be done by running a coin-tossing protocol [114] to generate a random value R_1. We suggest using SCRAPE [114], due to its low computational complexity and compatibility with public ledgers. Any solution that instantiates a publicly verifiable secret sharing (PVSS) scheme, however, would also be suitable. The only requirement is that the PVSS should output a value that is of the same type as the value output by the VRF function G.

For each subsequent round, participants then verify whether or not they are eligible to fold their randomness into the global value by checking if $H(G_{sk}(R_{\mathsf{rnd}})) <$ target, where the value of target depends on the number of expected leaders per round (for example, we choose $\mathsf{target} = H_{\mathsf{max}}/n_{\mathsf{rnd}}$ in order to have on expectation one leader per round). If this holds, then they reveal $y_{\mathsf{rnd}} = G_{sk}(R_{\mathsf{rnd}})$ and $p_{\mathsf{rnd}} = p_{sk}(R_{\mathsf{rnd}})$ to the other participants, who can verify that the inequality holds and that $\mathsf{Verify}_{\mathrm{VRF}}(R_{\mathsf{rnd}}, y_{\mathsf{rnd}}, p_{\mathsf{rnd}}) = 1$. If the participant is in fact eligible, then they are deemed to be the leader for that round and the global randomness is updated as $R_{\mathsf{rnd}+1} \leftarrow y_{\mathsf{rnd}}$.

To fully achieve security, we describe two necessary alterations to the basic protocol as presented thus far. First, in order to maintain unpredictability, participants should become eligible only after some fixed number of rounds have passed since they ran Commit. This means updating Verify to also check that $\mathsf{rnd}_{\mathsf{joined}} > \mathsf{rnd} - x$ (where $\mathsf{rnd}_{\mathsf{joined}}$ is the round in which the participant broadcast $\mathsf{tx}_{\mathsf{com}}$ and x is the required number of interim rounds). This has the effect that an adversary controlling f_B participants cannot privately predict that they will be elected leader for a few rounds and then grind through potential new secret key values to continue their advantage via new commitments, as the probability will be

sufficiently high that at least one honest participant will be elected leader between the time they commit and the time they participate.

Second, it could be the case that in some round, no participant is elected leader. It could also be the case that an adversary is the only elected leader and does not reveal their proof of eligibility and abort. To maintain liveness, we alter the protocol so that if no participant reveals $\mathsf{tx}_{\mathsf{rev}}$, we "update" the random beacon as $R_{\mathsf{rnd}} \leftarrow F(R_{\mathsf{rnd}})$, where F is a deterministic function that acts as a *Verifiable Delay Function* [115, 50]. When an honest player is not eligible on the winning block, they start computing the VDF and if by the time it is computed no leader has been revealed, they check their eligibility with the updated beacon $F(R_{\mathsf{rnd}})$. One can think of this as re-drawing the lottery after some delay. This allows participants to continue the protocol (and, since it is purely deterministic, is different from proof-of-work). In this study we will ignore the cost of running the VDF in our incentive analysis (as well as the cost of running the rest of the protocol). We consider this cost to be negligible compared to the rewards associated with being part of the protocol that we present in the next section. This is a limitation of Fantômette and we leave a deeper analysis including the price of running the VDF as an open problem. Moreover we make the assumptions that the time δ that it takes to compute the VDF is much bigger than the parameter Δ of our semi-synchronous model.

It could also be the case, however, that there are two or more winners in a round. In a setting such as proof-of-stake, being elected leader comes with a financial reward, and conflicts may arise if two winners are elected (such as the nothing-at-stake problem discussed in Section 2.2.3). One potential solution (also suggested by Algorand [38]) for electing a single leader is to require all participants to submit their winning values y_{rnd} and then select as the winner the participant whose pre-image y_{rnd} has the lowest bit value. This problem is investigated in Section 4.4, where we present the full Fantômette protocol. In this section, we simply allow for multiple leaders to be elected at the same round.

Finally, we describe a third, optional alteration designed to improve fairness by forcing more rotation amongst the leaders. To be elected, we require that a

participant has not acted as leader in the past $(n_{\mathsf{rnd}} - 1)/2$ rounds, which can be implemented by adding a condition in the Verify function and using $(n_{\mathsf{rnd}} + 1)/2$ in place of n_{rnd} in the computation of the value target.

Value of target We have chosen target such that one leader is elected on expectation. However this value could be changed to have more leaders elected on expectation if we want to rely on the VDF less often for example.

Accounting for a non-flat model By setting $\mathsf{target} = H_{\mathsf{max}}/n_{\mathsf{rnd}}$, each participants has a probability $1/n$ of being elected. The probability of being elected leader for a coalition that comprises n_C participants is, thus, $1 - (1 - \frac{1}{n})^{n_C}$ as they are *tossing a coin* for each of the participants in the coalition. In order to have a non-flat model, i.e., where a participant could own more than one fraction of stake, we could simply replace $\mathsf{target} = H_{\mathsf{max}}/n_{\mathsf{rnd}}$ by $\mathsf{target} = H_{\mathsf{max}} \times (1 - (1 - \frac{1}{n})^{n_C})$, where n_C is the amount owned by the participant and n is the total amount staked by all participants. This means that here the parameter target will be different for every participant. This change would not affect our protocol (whether Caucus or Fantômette), so we stick to the flat model for ease of exposition.

4.3.2 Security

In order to argue the security of Caucus as a whole, we first argue the security of its implicit random beacon. We note that because there are no incentives within Caucus, here we simply consider altruistic and Byzantine players (as is the case with traditional cryptography).

In terms of the fraction f_B of malicious participants that we can tolerate, as long as unpredictability and liveness are concerned, it is t/n, where t is the threshold of the PVSS scheme used to initialize the random beacon. As explained before, however, unbiasability of the random beacon is harder to argue for when the adversary controls many players as they could choose which of their winning shares to reveal or to hide. We investigate this in the next section, where we present the full Fantômette protocol. However, one key point here is that even if an adversary biases the leader election and random beacon by choosing which of their shares to reveal, the other

players still have a uniform probability of being elected leader and thus the leader election still satisfies our notion of fairness. Even if the adversary chooses the random beacon that they prefer, this does not reduce the probability of another player being elected leader (our leader election protocol accepts more than one leader). Thus we argue unbiasability of the random beacon in the case where the adversary controls one participant but argue for fairness even if an adversary controls t participants and is able to choose its prefered value of the random beacon.

Lemma 4.3.1. *If H is a random oracle and R is initialized using a secure coin-tossing protocol, then the random beacon R_{rnd} is also secure; i.e., it satisfies liveness (Definition 4.1.1) and unbiasability (Definition 4.1.3) with $f_B = t/n$, and unpredictability (Definition 4.1.2) with $f_B = 1/n$ for every subsequent round.*

Argument. For liveness, we observe that after initialization, no coordination is required, so any online participant can communicate their own eligibility to other online participants, allowing them to compute the new random value. The exception is the case where no participant is elected leader, in which case participants can update their random value by $R_{\mathsf{rnd}} \leftarrow F(R_{\mathsf{rnd}})$ until a leader reveals themselves.

For unpredictability, we must show that, unless the adversary is itself the next leader, it is hard to learn the value of R_{rnd} before it receives $\mathsf{tx}_{\mathsf{rev}}$. We have $R_{\mathsf{rnd}} = y_{\mathsf{rnd}-1} = G_{sk}(R_{\mathsf{rnd}-1})$, where $R_{\mathsf{rnd}-1}$ is assumed to be known. In the protocol, the adversary sees pk as part of the commitment of the relevant honest participant, and if that participant has run Reveal before it may have also seen $y'_{\mathsf{rnd}} = G_{sk}(R'_{\mathsf{rnd}})$ for $\mathsf{rnd}' < \mathsf{rnd}$. If the adversary could produce better than a random guess about R_{rnd} then they would also produce better than a random guess about $G_{sk}(R_{\mathsf{rnd}})$ which contradicts its pseudo-randomness.

For unbiasability, we need to show that R_{rnd} is computationally indistinguishable from a random string. By the assumption that R_{rnd} is unpredictable, as shown previously, and thus unknown at the time an adversary commits to their secret key, $G_{sk}(R_{\mathsf{rnd}})$ is also computationally indistinguishable from a random string due to the pseudo-randomness property of G (even under maliciously generated keys) and the fact that the adversary could not have grind through secret keys to bias

$G_{sk}(R_{\mathsf{rnd}})$. □

Theorem 4.3.2. *If H is a random oracle and R is initialized as a uniformly random value, then Caucus is a secure leader election protocol; i.e., it satisfies liveness, fairness, delayed unpredictability (where* barrier *is the step at which the elected leader reveals their proof), and private unpredictability (where* barrier *is the step at which the randomness R_{rnd} is fixed) with $f_B = t/n$.*

Argument. For liveness, a participant is elected if they broadcast a valid transaction $\mathsf{tx}_{\mathsf{rev}}$ such that $H(y_{\mathsf{rnd}}) < \mathsf{target}$. If `Update` satisfies f_B-liveness then an adversary controlling f_B participants cannot prevent honest participants from agreeing on R_{rnd}. In the case where no participants produce a value y_{rnd} such that $H(y_{\mathsf{rnd}}) < \mathsf{target}$, we update the value of R_{rnd} as described above until one participant is elected. Similarly as in the proof of Lemma 4.3.1, the protocol thus achieves liveness.

For fairness, a participant wins if $H(G_{sk}(R_{\mathsf{rnd}})) < \mathsf{target}_{\mathsf{rnd}}$. In the case where the adversary was not elected leader in the previous round, R_{rnd} is unbiasable (with the same argument used in the previous theorem) and thus uniformly distributed; we can thus argue that $y_{\mathsf{rnd}} = G_{sk}(R_{\mathsf{rnd}})$ is uniformly distributed. This implies that the probability that the winning condition holds is also uniformly random, as desired.

In the case where an adversary could choose which of their winning shares to reveal, an honest user still has probability $1/n$ of being elected leader. To see this, let's assume that the adversary has a choice between random beacons $R_{a1}, \cdots, R_{an}$. If the adversary wants to make sure that an honest leader does not have a probability $1/n$ of being elected leader, they need to choose R_{ai} such that for that participants $H(G_{sk}(R_{ai}))$ is not uniformly distributed (for example such that $H(G_{sk}(R_{ai})) > \mathsf{target}$ with probability more than $1 - 1/n$). By the unpredictability of the hash (that we model as a random oracle), the adversary cannot predict any such property unless it knows $G_{sk}(R_{ai})$ which then contradicts the pseudorandomness of G, as the adversary does not know the secret keys of the honest users. Thus the protocol is fair even in this case.

The argument for delayed unpredictability is almost identical to the one in Theorem 4.3.1: even when R_{rnd} is known, if $\mathcal{A}$ has not formed y_{rnd} itself then by the

pseudo-randomness of G it cannot predict whether $H(G_{sk}(R_{\mathsf{rnd}})) < \mathsf{target}_{\mathsf{rnd}}$. (If it could then the adversary could use that to distinguish $G_{sk}(R_{\mathsf{rnd}})$ from random with an advantage.) Finally, private unpredictability follows from the unpredictability of R_{rnd}. □

Even if the initial value was some constant instead of a randomly generated one, we could still argue that the protocol is fair after the point that the randomness of at least one honest participant is incorporated into the beacon. This assumption is weakened the longer the beacon is live, so works especially well in settings where the leader election protocol is used to bootstrap from one form of consensus (e.g., PoW) to another (e.g., PoS), as discussed for Ethereum [51].

4.3.3 Grindability

As already pointed out at the beginning of this section, Caucus is vulnerable to a form of *grinding*. An adversary controlling many eligible players could choose which player will reveal their proof of eligibility, and thus bias the outcome of the beacon.

One way of dealing with this problem, which has been presented in Ouroboros Praos [46], is to have one random beacon per *epoch*, i.e., one beacon used for a specified number of consecutive rounds instead of having a new random beacon at each round.

As in Caucus, players still output one value R_{rnd} per round but now the beacon that they use to check their eligibility is a value $\mathcal{R}_{\text{epoch}}$, which is computed as $\mathcal{R}_{\text{epoch}} = R_i||R_{i+1}||\cdots||R_j$. The values (i, j) correspond to rounds far in the past chosen such that the $j-i+1$ values that are concatenated ensure that each beacon $\mathcal{R}_{\text{epoch}}$ includes at least one random number R_{i+k} from an honest participant. An adversary that chooses which of their own values to reveal would thus have less impact on the final value of the beacon $\mathcal{R}_{\text{epoch}}$. Furthermore, since the value $\mathcal{R}_{\text{epoch}}$ is used for many rounds (the length of the epoch) the adversary is less likely to find a beacon that would give them an advantage on many consecutive round. The longer the epoch is, the less impact the grinding will have on the adversary's future eligibility.

This solution could be applied to Fantômette in order to provide stronger guarantees against grinding, but this would mean losing the private unpredictability property. If the random beacon is fixed for many rounds, any player is able to predict well in advance when they are going to be elected leader and disconnect the rest of the time. Therefore, we instead consider the simpler random beacon Caucus presented above. This means that the protocol is more vulnerable to grinding, which we deal with at the price of considering a more limited adversary than the one in Ouroboros Praos, while maintaining the desirable property of private unpredictability.

Dealing with grindability for a non-specific adversary while maintaining private unpredictability is an open problem.

4.4 Fantômette: A Consensus Protocol

In this section, we present Fantômette, our full blockchain consensus protocol. Our focus is on incentives, and in particular on enforcing good behavior even in settings where no natural incentives or investments exist already.

Briefly, Fantômette works as follows: participants bet on the block that has the strongest *score*, according to their view of the blockDAG. They also reference all the leaves (i.e., the most recent blocks) of which they are aware, to "prove" that they are well connected and following the rules. A block is valid if among all of its references, it is indeed betting on the one with the higher score. We argue for its security more extensively in the next section, but give here some intuition for how it addresses the challenges presented in the PoS setting introduced in Section 2.2.3. First, Fantômette solves the nothing-at-stake problem by strongly punishing players who do not reference their own blocks, and ensuring that players bet only on the strongest chain they see. As it is not possible for two chains to appear as the strongest at the same time, this prevents players from betting on multiple chains.

The grinding attack is quantified according to a specific adversary that we detail in Section 4.5.1. A more general adversary could be considered, with a trade-off, as discussed in Section 4.3.3, but we leave this analysis as an open problem.

Finally, long-range attacks are thwarted by Fantômette's finality rule, which acts as a form of decentralized checkpointing.

4.4.1 Protocol specification

We specify how to instantiate the algorithms required for a consensus protocol specified in Section 4.2.3. The Setup and Eligible algorithms are as described for Caucus in Section 4.3. Before presenting the rest of the algorithms, we give some additional definitions associated with finality in blockDAGs.

In order to make the following definitions, we assume a static set of players. We then present how to handle a dynamic set. Let's also recall that thanks to the deposit, the set of players is known to everyone. A *candidate* block is a block, as its name suggests, that is a candidate for finality. The initial list of candidate blocks consists of every block *betting* on the genesis block; i.e., every block that uses this as its parent $\mathsf{B}_{\mathsf{prev}}$. We say that a block B received two-thirds of bets if two-thirds of the players have bet on block B. More formally, we say that block B received two-thirds of bets if

$$|\{\mathsf{B}'.\mathsf{snd}\ ;\ \mathsf{B}' \text{ such that } \mathsf{B} \in \mathsf{Ancestors}(\mathsf{B}')\}| > 2/3|\mathcal{P}|$$

Whenever a block has in its past two-thirds of bets on a candidate, this block acts as a *witness* to the finality of that block, so is called a witness block. More formally block B_1 is a witness for B_0 if:

$$|\{\mathsf{B}_i.\mathsf{snd}\ ;\mathsf{B}_i \in \mathsf{Past}(\mathsf{B}_1) \text{ and } \mathsf{B}_0 \in \mathsf{Ancestors}(\mathsf{B}_1)\}| > 2/3|\mathcal{P}|$$

If a block B_1 is a witness for B_0 and B_2 is a witness for B_1, we say that B_2 is an *attestor* of B_0. Finally, candidate blocks belong to a given *rank*, which we denote by rk. The first set of candidate blocks (after the genesis block) belong to $\mathsf{rk} = 1$. After this every block that bets on an attestor block of rank rk and has a distance of w with this block is a candidate for rank $\mathsf{rk}+1$. The above process constitutes a decentralized checkpointing. The parameter w represents a period where the checkpointing is paused in order to handle a dynamic set of players. Once a block is

finalized (i.e. once there exists at least one attestor block) we allow for a window of w blocks for the players to leave or join the protocol. At the end of this period, the set of players is fixed and the decentralized checkpointing resumes with, as new candidate blocks, all blocks that have a distance w with an attestor block for rank rk. We leave as important future work a solution that would allow players to leave and join the protocol even during the checkpointing.

Using the sample blockDAG in Figure 4.1, for example, and assuming four players and $\mathsf{w} = 0$, the initial list of candidate blocks is $\{A, B, C, D\}$ (which all have rank $\mathsf{rk} = 1$). Block E then bets on A, as does block H (since A is an ancestor of H), so, assuming that block A, E and H were all created by different players, H can be considered as a witness block for A. Similarly, I acts as a witness block for C (assuming that C, G and I were all created by different players). Since this now constitutes two-thirds of the participants, A and C become *justified* [51]. If in turn two-thirds of participants place bets on the associated witness blocks, then the justified block becomes *finalized*. With $\mathsf{w} = 0$, the attestor blocks are added to the candidate list accordingly, but at rank $\mathsf{rk} = 2$.

Fork choice rule We present a formal specification of our fork choice rule FCR in Algorithm 4.4.1. Intuitively, the algorithm chooses blocks with more connections to other blocks. Accordingly, we compute the score of a leaf block B by counting the number of outgoing references for every block in its past. We do not count, however, blocks that have been created by an adversary using the same eligibility proof multiple times. We denote as Double the set of all blocks that contains the same proof of eligibility but different content. The score of a chain whose tip is B is then the number of edges in the subgraph induced by $\mathsf{Past}(\mathsf{B}) \setminus \mathsf{Double}$, and we pick as a "winner" the leaf with the highest score. If there is a tie (i.e., two leaf blocks have the same score), we break it by using the block with the smallest hash.[2]

Betting To place a bet, a participant first identifies the latest winning block as $\mathsf{B} \leftarrow \mathsf{FCR}(\mathsf{G}_{\mathsf{pub}})$. They then check their latest attestor block (if they have one) and verify

[2]It is important, to avoid grinding attacks, to use the hash as defined in Caucus, or something else similarly unbiasable.

Algorithm 4.4.1: Fork-choice rule (FCR)

input : a DAG G

output: a block B representing the latest "winner"

if $(\mathsf{G} = \textit{Genesis Block})$ **then**
  return G

$w \leftarrow \emptyset$

for $\mathsf{B} \in \mathsf{Leaves}(\mathsf{G})$ **do**
  $w[\mathsf{B}] = |\mathsf{B}[\mathcal{B}_{\mathsf{leaf}}]|$
  for $\mathsf{B}' \in \mathsf{Past}(\mathsf{B}) \setminus \mathsf{Double}$ **do**
    $w[\mathsf{B}] += |\mathsf{B}'[\mathcal{B}_{\mathsf{leaf}}]|$

$\mathsf{CW} \leftarrow \mathrm{argmax}_{\mathsf{B} \in \mathsf{Leaves}(\mathsf{G})} w[\mathsf{B}]$

```
// if there is a tie choose block with smaller hash
```

$\mathsf{B} \leftarrow \mathrm{argmin}_{\mathsf{B} \in \mathsf{CW}} H(\mathsf{B})$

return B

that at least one candidate block associated with it is also in $\mathsf{Ancestors}(\mathsf{B})$ (i.e. they verify that the block was not created maliciously as part of a long range attack). They then check to see if they are eligible to act as a leader by computing $\pi \xleftarrow{\$} \mathsf{Eligible}(\mathsf{B}, \mathrm{st}_{\mathsf{priv}})$. If they are (i.e., if $\pi \neq \perp$), then they form a block with B as the parent, with all other blocks of which they are aware as the leaf blocks, and with their proof of eligibility π and set of transactions.

Block validity We now define the rules that make a block valid; i.e., the checks performed by $\mathsf{VerifyBlock}(\mathsf{G}, \mathsf{B})$. Intuitively, a valid block must be betting on the block chosen by the fork-choice rule, and its creator must be eligible to bet on that block. If a player is aware of a justified block, then they must bet on either that block or another witness block, but cannot prefer a non-justified block to a justified one.

More formally, a new block $\mathsf{B} = (\mathsf{B}_{\mathsf{prev}}, \mathcal{B}_{\mathsf{leaf}}, \pi, \mathsf{txset})$ is valid only if the following hold:

1. The creator is eligible to bet: $\mathsf{Eligible}(\mathsf{B}_{\mathsf{prev}}, \mathrm{st}_{\mathsf{priv}}^{\mathsf{B.snd}}) \neq \perp$.

2. It is betting on the block chosen by the fork choice rule for the blocks of which it is aware; i.e., $\mathsf{B}_{\mathsf{prev}} = \mathsf{FCR}(\mathsf{Past}(\mathsf{B}))$.

3. If it references a witness block then it is betting on a witness block; i.e., if there exists a witness block in $\mathsf{Past}(\mathsf{B})$ then there exists a witness block in $\mathsf{Ancestors}(\mathsf{B})$. Intuitively, this means that if a player is aware of a justified block, they cannot prefer a non-justified block over a justified block.

4. If it references an attestor block, then it is betting on a block in the past of that block: if there exists a second witness block $\mathsf{B}_s \in \mathsf{Past}(\mathsf{B})$, then $\mathsf{Ancestors}(\mathsf{B}) \cap \mathsf{Past}(\mathsf{B}_s) \neq \emptyset$. We will prove in section 4.5.2 that once an attestor block exists, all the altruistic players agree on the set of candidate blocks, so this ensures that this set does not grow (i.e., if a player reveals their block too late in the future, after an attestor block already exists for the candidate blocks at that height, this block will not be added to the chain).

4.4.2 Incentives

Label We present a specification of our Label function in Algorithm 4.4.2. Intuitively, if a block is chosen by the FCR it is labelled winner and so are all of its ancestors. Blocks that bet on winners are labeled neutral. Following the techniques in PHANTOM [56], all winning and neutral blocks form a subset of the DAG called the *blue* subset, denoted by blue. Every block whose anticone intersects with fewer than k blocks in the blue set is labeled neutral, and otherwise it is labeled loser. The parameter k is called the *inter-connectivity* parameter, and means that a block is allowed to be "unaware" of k winning blocks, but not more (as, e.g., these blocks may have been created at roughly the same time).

Utility functions At the end of the game, which is of length T, as defined in Section 4.2, we take the BCPD and apply the Label function to it, in order to associate each block with a state. For every winning block that a player has added to the BCPD, they win a reward of $\mathsf{rwd}(\mathsf{B})$, and for every losing block they lose pun. In addition, if a player creates a block that does not reference one of their own blocks, we add a bigger punishment bigpun (for example, blocks that belong to the set Double defined previously will add this punishment). This punishment is bigger because a player is obviously aware of all of their own blocks, so if they do not

Algorithm 4.4.2: Label

input : A DAG G

output: A labelling of the blocks in the DAG M

set B $\leftarrow$ FCR(G)

blue $\leftarrow \{B\}$

$M(B) =$ winner

for $B_i \in$ Ancestors(B) **do**
    blue $\leftarrow$ blue $\cup \{B_i\}$

    $M(B_i) =$ winner

    for $B_j \in$ DirectFuture(B_i) $\setminus$ Ancestors(B) **do**
        blue $\leftarrow$ blue $\cup \{B_j\}$

        $M(B_j) =$ neutral

for $B_i \in G \setminus$ blue **do**

    `// Here we assume that the blocks in G are ordered by increasing height. If there are multiple blocks at one height, they are ordered by increasing hash.`

    if $|\text{Anticone}(B_i) \cap \text{blue}| \leq k$ **then**
        blue $\leftarrow$ blue $\cup \{B_i\}$

        $M(B_i) =$ neutral

    else
        $M(B_i) =$ loser

return M

reference one it is an obvious form of misbehavior (whereas a block might end up being labelled a loser for other reasons).

More formally, we define the following utility function:

$$u_i(\mathsf{BCPD}) = \sum_{\substack{\mathsf{B}\in\mathsf{BCPD}\text{ s.t.}\\ \mathsf{B.snd}=i\text{ and}\\ M(\mathsf{B})=\text{winner}}} \mathsf{rwd}(\mathsf{B}) - \Big(\sum_{\substack{\mathsf{B}\in\mathsf{BCPD}\text{ s.t.}\\ \mathsf{B.snd}=i\text{ and}\\ M(\mathsf{B})=\text{loser}}} \mathsf{pun} + \mathsf{bigpun} \times |N_i| \Big) \quad (4.1)$$

where $M = \mathsf{Label}(\mathsf{BCPD})$ and $N_i = \{(\mathsf{B}_j, \mathsf{B}_k) \text{ s.t. } \mathsf{B}_j.\mathsf{snd} = i \wedge \mathsf{B}_k.\mathsf{snd} = i \wedge \mathsf{B}_j \notin \mathsf{Past}(\mathsf{B}_k) \wedge \mathsf{B}_k \notin \mathsf{Past}(\mathsf{B}_j)\}$.

The reward function is proportional to the connectivity of a block; i.e., a block that references many blocks receives more than a block that references only one other block. The reason is that we want to incentivize players to exchange blocks between each other, rather than produce blocks privately (as in a selfish mining attack). In this paper we consider a simple function $\mathsf{rwd}(\mathsf{B}) = |\mathcal{B}_{\mathsf{leaf}}| \times c$ for some constant c, and treat pun and bigpun as constants. We leave the study of more complex reward and punishment mechanisms as an interesting open problem.

One of the difficulties of dealing with blockchain-based consensus, compared to traditional protocols, is that the enforcement of the payoff is achieved only by consensus; i.e., the utilities depend on whether or not enough players enforce them. In order to enforce the payoff, we thus assume that participants can give a reward to themselves in forming their blocks (similarly to Bitcoin), but that evidence of fraud can be submitted by other players. If another player submits evidence of fraud, the subsequent punishment is taken from the security deposit of the cheating player.

4.4.3 Compound effect

A phenomenon discussed in the context of proof-of-stake is the "rich get richer" effect [116]. The intuition is that if participants are rewarded proportionally to their stake, someone richer will be rewarded more often, and thus will be able to reinvest their stake to be rewarded even more often. This is an undesirable effect, as ideally we would like the proportion of stake to stay constant over time for a fixed set of participants. In Fantômette, this compounding effect is mitigated because players need to wait before being able to reinvest their stake in the betting protocol. This is

due to the limitations introduced in Section 4.3 where players have to wait before depositing or taking out stake. This way the reward is not immediately reintroduced and this leaves enough time such that on average, the rewards received will be fairly distributed. We omit the full proof of this and leave it as an open problem.

4.5 Security of Fantômette

In this section, we start by defining a type of adversary (Section 4.5.1) and we argue that Fantômette is secure against this adversary, according to the model defined in Section 4.2. We support our arguments with a simulation of the protocol as a game played between Byzantine, rational and altruistic players in Section 4.5.3 where Byzantine and rational players follow our adversary specification. In this section, to ease the notation, we write p the probability for one player to be elected leader (for example when choosing one leader elected on expectation we have $p = 1/|\mathcal{P}| = 1/n$) and $n_{\mathcal{C}}$ the number of players controlled by the adversary ($n_{\mathcal{C}} = \lfloor f_B \times n \rfloor$).

4.5.1 Adversary Specification

We first discuss the different strategies available to a coalition of players, whether Byzantine or rational. They can take any deviation possible from the game. We do assume, however, that they create valid blocks, as otherwise passive players will simply ignore their chains (as discussed in Section 4.2).

If an adversary withholds its blocks, it can gain an advantage in subsequent leader elections. To see this, consider that after each block each player has a probability $1/n$ of being elected leader. Being a leader does not guarantee a winning block, however, as only the block with the smallest hash wins. For each block, the number of subsequent players k that are elected leader follows a binomial distribution parameterized by n and $1/n$. Assuming the leader election is secure, each of these leaders is equally likely to have the winning block, so each player has probability $1/(kn)$ of being the winner. By not revealing a block, this probability goes up to $1/n$ for each player controlled by the adversary (since other players are simply not aware of it, they cannot verify their eligibility and create blocks that would compete with the adversary). By keeping their chain private an adversary can raise their

chance of having a winning block. We thus assume that both Byzantine and rational players withhold their blocks and grind through all possible subsequent blocks in order to maximize their advantage.

In terms of the space that players grind through, the main option they have when elected leader is whether to place a bet or not. The protocol dictates that they must bet on the fork-choice rule only, but they may wish to bet on a different block (doing so could increase their chances of being elected leader in the future). In order to maintain the validity of their blocks, they must eliminate references in their set $\mathcal{B}_{\mathsf{leaf}}$ so that their chosen block appears as the fork-choice rule (in accordance with the first check in VerifyBlock). Players are, however, always better off including as many blocks as possible in their references, as it increases the score of their block. Thus, they remove enough references to blocks that have higher scores to make their block appear as the fork-choice rule, but not more.

For rational players, there is a trade-off between not revealing their block (which raises their chance of having more winning blocks, as argued above) and revealing their block, which reduces their chance of having more winning blocks but increases their reward because it allows their block to have more references. Our simulation investigates this trade-off. Regardless, the strategy of rational players is to grind through all possible blocks and broadcast the chain that maximizes their utility. Byzantine players, in contrast, do not try to maximize their profit, but instead play irrationally, i.e., they are not deterred by punishment. In both cases, Byzantine and rational, the grinding stops whenever the players in the coalition are not elected leaders. We do not consider the case where players use the VDF function sequentially until they are elected leader again since doing so will result in a attack that takes a long time. As we will explain in our security arguments, our decentralized checkpointing mechanism prevents long forks from happening; this motivates this additional assumption although it limits the generality of our model.

To summarize, our Byzantine adversary follows the below strategy. Whenever it is elected leader, it grinds through all the possible valid blocks it can create and stops whenever it is not elected leader anymore without running the VDF. More

precisely, the grinding works as follows: the adversary checks which of its players is elected leader in the first round of grinding. If there is at least one player elected, then the adversary creates a block for each elected player, without broadcasting it to the other players and continues grinding. For each block created in the first round of grinding, the adversary will check again if one of its players is eligible to create a block and if it is, it will do so, without broadcasting it and so on and so forth. The adversary thus grows multiple subDAGs at the same time, as long as it can. Since the randomness on each block created will be different, according to our leader election, each subchain or subDAG will potentially have a different number of blocks. When the adversary no longer has a player elected, it stops grinding (as long as the adversary controls a limited fraction of the participants this will happen quickly, as we will see in Theorem 4.5.3). We then look at the worst possible subDAG created, according to the security property considered, and assume that the adversary broadcasts this chain to the rest of the players. For example, for immunity the adversary chooses to broadcast the subDAG with the most blocks, as this is the most likely to harm honest players (we will detail why in the argument for $t-$immunity). For convergence, the adversary chooses to broadcast the subDAG with the most references as this is the one that is the most likely to make honest players switch chains (as honest players choose the heaviest chain). For chain quality as we are only concerned with the blocks that are included in the main chain, we assume that the adversary broadcasts the sub-DAG that contains the longest chain but assume that this subDAG will always be included in the chain, regardless of its score. A rational coalition will operate in the same way, except they will broadcast the sub-DAG that maximize their revenue.

In both cases we assume this grinding stops whenever the coalition is not elected leader anymore without using the VDF as specified in Section 4.3 and thus the adversary broadcasts its subDAG as soon as it is not elected leader anymore on any of the blocks it has created as part of the grinding. Once the adversary has broadcast its chain, honest miners can start mining on it, and as soon as the adversary is eligible again, they will go on and grind again. The above adversary is quite

limited and specific. Specifically, we consider a *short-term* adversary that stops its grinding attack when having to run the VDF, although we also consider the specific long-range attack discussed in Section 2.2.3. Considering a more general *medium-term* adversary would require to trade-off the private unpredictability of the random beacon, to some extent, as discussed in Section 4.3.3. In this thesis, we made the choice of limiting our adversary, instead, and investigate the security achieved under this limited grinding adversary. We leave as an open problem the study of security under a more general class of adversaries and we believe that the methods used in this thesis could be helpful in doing so.

4.5.2 Security arguments

According to the security properties in Section 4.2.3, we need to argue three things: convergence, chain quality, and robustness. We support these security properties with our simulation in Section 4.5.3. We leave formal proofs as an open problem. All of the security arguments and simulations stand with respect to the Byzantine and rational behaviours detailed in the previous section, unless specified otherwise.

Before arguing each security property, we first argue some general results about the protocol. We call the strongest chain the chain with the highest score according to a hypothetical oracle node that collates the views of the blockDAG maintained by all participants.

Claim 4.5.1. *For any altruistic, rational or Byzantine players following their respective strategies defined in Section 4.5.1, a block added to the strongest chain will have a higher score than a block added on a weaker chain.*

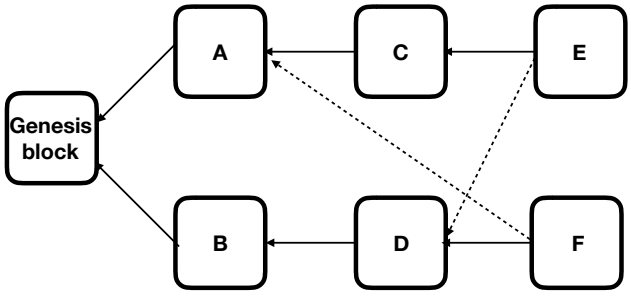


Figure 4.3: Example of two blocks added on competing chains.

Argument. As a reminder, the score of a chain whose tip is B is the number of edges in $\mathsf{Past}(B)$. Since the blocks that have the same leader at the same epoch (denoted previously as the set Double) do not count for the score, we simply ignore them for this proof. We consider two blocks B_1 and B_2 that are the tips of two chains C_1 and C_2 of scores, respectively, S_1 and S_2. We assume that C_1 is the stronger one, i.e., B_1 is the block chosen by the fork choice rule. This implies that $S_1 \geq S_2$.

A player that is creating a block betting on block B_1 will include all the blocks they are aware of. This is true whether the player is altruistic or not as (as specified in section 4.5.1) a rational or Byzantine player will always include all the leaves that they can when creating a block. On the other hand, a player creating a block betting on B_2 will have to ignore at least block B_1 in order to create a valid block (according to our block validity rules). This is, intuitively, why we get our result (see Figure 4.3 for a visual representation). Below we detail the argument in the case of exactly two chains to choose from and explain how it extends to the case of multiple chains.

We consider two different cases: the one where $\mathsf{B}_1[\mathcal{B}_{\mathsf{leaf}}] = \mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$ (i.e., B_1 and B_2 reference the same set of blocks) and one where $\mathsf{B}_1[\mathcal{B}_{\mathsf{leaf}}] \neq \mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$. We note B'_1 and B'_2 the blocks betting on B_1 and B_2 respectively.

First case: $\mathsf{B}_1[\mathcal{B}_{\mathsf{leaf}}] = \mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$**.**
In this case it is straightforward that $S_1 = S_2$ (since $\mathsf{Past}(\mathsf{B}_1)$ and $\mathsf{Past}(\mathsf{B}_2)$ have the same number of edges). Additionally, B'_1 references both B_1 and B_2 while B'_2 cannot reference B_1. The number of edges in $\mathsf{Past}(\mathsf{B}'_1)$ is thus the sum of (1) the number of edges in $\mathsf{Past}(\mathsf{B}_1)$ (which is S_1 by definition); (2) $\mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$ since B'_1 references B_2 all of the reference from B_2 to $\mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$ are also included in $\mathsf{Past}(\mathsf{B}'_1)$; (3) two since B'_1 adds two edges by referencing both B_1 and B_2. Hence the score of B_1 will be: $S'_1 = S_1 + 2 + \mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$.

B'_2 on the other hand is not referencing B_1 so its score is $S'_2 = S_2 + 1 = S_1 + 1$, since $S_1 = S_2$. It is thus clear that B_1 has a higher score than B_2.

Second case: $\mathsf{B}_1[\mathcal{B}_{\mathsf{leaf}}] \neq \mathsf{B}_2[\mathcal{B}_{\mathsf{leaf}}]$**.**
The number of edges in $\mathsf{Past}(\mathsf{B}'_1)$ is the sum of (1) the number of edges in

$\mathsf{Past}(\mathsf{B}_1) \bigcup \mathsf{Past}(\mathsf{B}_2)$ (2) two because B'_1 adds two edges by including both B_1 and B_2. Thus the score of B'_1 is $S'_1 = |\mathsf{Edges}(\mathsf{Past}(\mathsf{B}_1) \bigcup \mathsf{Past}(\mathsf{B}_2))| + 2$.

B'_2 on the other hand is not referencing B_1. Additionally, it could be the case that $S_1 > S_2$ and hence B'_2 may need to ignore more than one block in order to be valid (according to our block validity rule 2). Since B'_2 is betting on B_2, the other blocks that it references must have a score of at most S_2. This means that B'_2 will trim $\mathsf{Past}(\mathsf{B}_1)$ in order to include only the blocks that have a score of at most S_2. The score of B'_2 is thus the sum of: (1) the sum of the edges in $\mathsf{Past}(\mathsf{B}_2) \bigcup \mathsf{Past}(\mathsf{B}_1)^{\text{trim}}$ where $\mathsf{Past}(\mathsf{B}_1)^{\text{trim}}$ is a strict subset of $\mathsf{Past}(\mathsf{B}_1)$; (2) one because B'_2 adds one more edge by referencing B_2 (the edges that B_2 add by referencing blocks in $\mathsf{Past}(\mathsf{B}_1)$ are counted in $\mathsf{Past}(\mathsf{B}_1)^{\text{trim}}$ since these blocks were already referenced in $\mathsf{Past}(\mathsf{B}_1)$). Thus the score of B'_2 is $S'_2 = |\mathsf{Edges}(\mathsf{Past}(\mathsf{B}_2) \bigcup \mathsf{Past}(\mathsf{B}_1)^{\text{trim}})| + 1$. Since $\mathsf{Past}(\mathsf{B}_2) \bigcup \mathsf{Past}(\mathsf{B}_1)^{\text{trim}}$ is strictly included in $\mathsf{Past}(\mathsf{B}_2) \bigcup \mathsf{Past}(\mathsf{B}_1)$, it is straightforward that B'_1 will have a higher weight than B'_2.

If there are $m > 2$ chains then the argument works the same way except that instead of having two edges added for B'_1 there will be m edges added (for referencing all the tips of the chains) and for B'_2 instead of having one edge added there will be, at most, $m-1$ as B'_2 has to ignore at least B_1 (and potentially more blocks if they have a score higher than B_2). Additionally, instead of considering $\mathsf{Past}(\mathsf{B}_1) \bigcup \mathsf{Past}(\mathsf{B}_2)$ for B'_1 we consider $\mathsf{Past}(\mathsf{B}_1) \bigcup \mathsf{Past}(\mathsf{B}_2) \bigcup \cdots \bigcup \mathsf{Past}(\mathsf{B}_m)$ and a strict subset of this union for B'_2.

□

We show in Theorem 4.5.2 that for a given rank, all attestors share the same set of candidate blocks provided that $f_B < 1/3$ (the rest of the players are altruistic). The idea is that whenever a block is finalized, all players have agreed on the current set of candidate blocks, and thus they should not accept any other candidate blocks at that rank (as required by the betting algorithm stated in Section 4.4). The following theorem considers a general adversary.

Theorem 4.5.2 (Finality)**.** *If $f_B < 1/3$ and $f_A = 1 - f_B$, then once a block at rank*

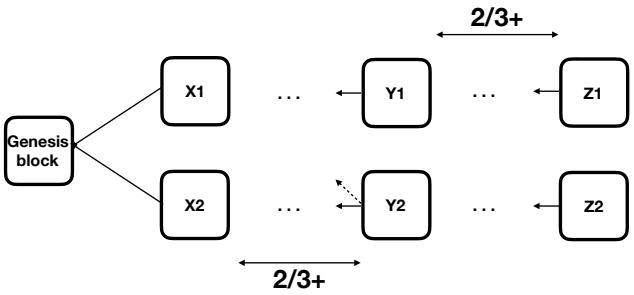


Figure 4.4: A visual sketch of the proof of Theorem 4.5.2. A participant placing a bet between x_2 and y_2, and y_1 and z_1 must have placed their bet on x_2 first, thus z_1 references x_2.

rk *is finalized, players agree on a list of candidate blocks for rank* rk*, i.e., this list cannot grow anymore.*

Argument. Assume there exist two finalized blocks x_1 and x_2. Denote by y_i the witness block for x_i, and by z_i the attestor block for x_i. By the definition of finality, it must have been the case that more than two-thirds of participants placed bets on x_1 and x_2, which in turn implies that more than a third of them placed bets on *both* x_1 and x_2. Since only a third of participants are Byzantine, this means that at least one non-Byzantine player placed a bet on both blocks. (Recall that during the decentralized checkpointing the set of players is fixed.) Altruistic players always reference their own block; this means that there is one of the witnesses blocks y_i that references "across the chains"; i.e., such that $x_1, x_2 \in \mathsf{Past}(y_i)$.

Without loss of generality, assume that $x_1, x_2 \in \mathsf{Past}(y_2)$. This means that $x_1 \in \mathsf{Past}(z_2)$ and thus $x_1, x_2 \in \mathsf{Past}(z_2)$. Since more than two-thirds of the participants bet on y_1, by a similar reasoning as above this means that at least one altruistic player placed a bet on both y_1 and x_2 (before y_2). Thus, either (1) z_1 references x_2 or (2) y_2 references y_1. Because of the third check in $\mathsf{VerifyBlock}$, however, this second case is not possible: since y_1 is a witness block, y_2 cannot reference it without betting on a justified block, and x_2 is not justified before y_2. We refer the reader to graph 4.4 for a visual intuition of this. Thus, it must be the case that z_1 references x_2. Thus z_i references both x_i and x_j for $i \neq j$.

For a set of candidate blocks, all the attestor blocks thus reference all candidate blocks (applying the previous analysis to all the candidate blocks pair-wise). So,

after a candidate block is finalized, players must agree on the set of candidate blocks for that rank. □

We now show our main theorems, starting with chain quality. A player is unlikely to be elected winner for many consecutive blocks and here we quantify the advantage that the adversary that we defined in Section 4.5.1 can gain by grinding. A reminder that since chain quality captures the maximum fraction of blocks that an adversary can contribute, without any notion of incentive, here we consider only a Byzantine adversary against altruistic players. This theorem depends on a security parameter $p_\alpha \in]0,1[$ to be chosen by protocol designers.

Theorem 4.5.3 (Chain quality)**.** *For every security parameter $p_\alpha \in]0,1[$, there exists $\mu > 0$ such that a coalition $f_B < 1/3$ of Byzantine adversaries that follows the strategy defined in Section 4.5.1 cannot contribute to a fraction of more than $1-\mu$ of the blocks in the main chain except with probability p_α (μ depends on the value of p_α).*

Argument. Recall that, as per the assumptions described in Section 4.5.1, we consider an adversary that tries to grind as many blocks as they can privately, stopping when they are no longer elected leaders (without using the VDF).

We now move on to compute the probability that the adversary can keep on grinding for ℓ consecutive rounds. In order to grind for ℓ consecutive rounds, the adversary needs to have one of its players winning at each height for ℓ consecutive rounds (otherwise grinding stops). We denote x_i the total number of successful leaders controlled by the adversary at each round i of grinding. In round 1 the adversary must have $1 \leq x_1 \leq n_{\mathcal{C}}$ leaders in order to continue grinding. This happens with probability $\sum_{x_1=1}^{n_{\mathcal{C}}} \binom{n_{\mathcal{C}}}{x_1} p^{x_1}(1-p)^{(n_{\mathcal{C}}-x_1)}$. (This follows from the Binomial distribution with parameter $n_{\mathcal{C}}$ and p.) Then in the second round the adversary will be able to toss $n_{\mathcal{C}}$ coins for each of the successes of round 1 thus the adversary tosses $n_{\mathcal{C}} \times x_1$ coins that each succeeds with probability p (as all the tosses are independent). On round 2, for a fixed x_1 the probability that the adversary has exactly $x_2 \geq 1$ successes is $\binom{x_1 n_{\mathcal{C}}}{x_2} p^{x_2}(1-p)^{(n_{\mathcal{C}} x_1 - x_2)}$ (following the Binomial distribution with parameter $n_{\mathcal{C}} \times x_1$ and p) and again to have the probability of two consecutive

successes we must sum the above probabilities for all values of $1 \leq x_1 \leq n_{\mathcal{C}}$ and $1 \leq x_2 \leq x_1 n_{\mathcal{C}}$. Then in round 3 the adversary can toss $n_{\mathcal{C}} \times x_2$ coins and so on and so forth. Summing over all the possible combination of $(x_1, \cdots, x_\ell)$ gives us that the probability of grinding for at least ℓ rounds is

$$\sum_{x_1,\ldots,x_\ell \in S_\ell} \binom{n_{\mathcal{C}}}{x_1} \ldots \binom{n_{\mathcal{C}} x_{\ell-1}}{x_\ell} p^{\sum_{i=1}^{\ell} x_i} (1-p)^{n_{\mathcal{C}}(1+\sum_{i=1}^{\ell-1} x_i) - \sum_{i=1}^{\ell} x_i}$$

where $S_\ell = \{x_1, \ldots, x_\ell : 1 \leq x_1 \leq n_{\mathcal{C}} \; ; \forall i > 1 \; 1 \leq x_i \leq x_{i-1} n_{\mathcal{C}}\}$.

For every ℓ, we note the above probability $\mathcal{P}_\ell$. It is straightforward that the probability of grinding for $\ell + 1$ consecutive rounds is strictly less than that of grinding for ℓ rounds. Succeeding in grinding for $\ell + 1$ rounds implies succeeding in grinding for ℓ rounds.

Since $\ell \mapsto \mathcal{P}_\ell$ is strictly decreasing and bounded from below by zero, it has a limit when $\ell \to \infty$. We show that this limit is equal to zero. For the sake of contradiction, we assume this limit to be strictly positive, which we denote λ_0. This means that for every ℓ, $\mathcal{P}_\ell > \lambda_0$ (since $\ell \mapsto \mathcal{P}_\ell$ is decreasing). Denote X the random variable capturing the exact number of rounds that the adversary grinds through. We have $\mathcal{P}_\ell = \Pr[X \geq \ell] = \sum_{i=\ell}^{\infty} \Pr[X = i]$. We thus have, for every ℓ: $\sum_{i=\ell}^{\infty} \Pr[X = i] > \lambda_0$. Additionally we have $\sum_{i=0}^{\infty} \Pr[X = i] = 1$, which implies that $1 - \sum_{i=0}^{\ell-1} \Pr[X = i] > \lambda_0$ for every ℓ. However since $\sum_{i=0}^{\infty} \Pr[X = i] = 1$, we have $1 - \sum_{i=0}^{\ell-1} \Pr[X = i] \to_{\ell \to \infty} 0$ and hence there exist some ℓ_0 such that for every $\ell \geq \ell_0$ we have: $0 \leq 1 - \sum_{i=0}^{\ell-1} \Pr[X = i] < \lambda_0$. This is a contradiction and hence it must be that $\lambda_0 = 0$.

Thus, for every $\lambda > 0$, there exists an ℓ_0 such that for every $\ell \geq \ell_0$:

$$\mathcal{P}_\ell < \lambda$$

An appropriate choice of λ ensures that the probability of grinding for ℓ rounds is small enough to assume that the grinding will stop after ℓ blocks.

It could be the case that there is more than one block on which the adversary can start grinding (e.g., if two honest leaders were elected at that round the adversary could start grinding on both blocks). Since each honest player creates at most one block at each round, the maximum number of honest blocks created at one round is

$n - n_{\mathcal{C}}$. Hence, following standard results on Binomial distribution with parameter $\mathcal{P}_\ell$ and $n - n_{\mathcal{C}}$, none of the potential $n - n_{\mathcal{C}}$ grinding opportunities in a round will last for longer than ℓ rounds except with probability $p_0 = 1 - (1 - \mathcal{P}_\ell)^{n-n_{\mathcal{C}}}$. We have that: $p_0 < 1 - (1 - \lambda)^{n-n_{\mathcal{C}}} \approx (n - n_{\mathcal{C}})\lambda$, for a small λ.

After the grinding stops, as per our assumption, the adversary broadcasts its chain to the honest players. We now consider a worst-case scenario where the grinded blocks broadcast by the adversary are always included in the main chain and where the adversary grinds at each round ℓ blocks. By definition, the adversary is not elected leader at the end of its grinded chain (since the adversary broadcasts its chain as soon as it is not elected leader). The honest players thus contribute the next block with probability $p_h = 1 - \Pr[\text{honest player have no leader}] = 1 - ((1 - p)^{n-n_{\mathcal{C}}})$, following standard results on Binomial distribution. If the honest players are not elected leaders then everyone will compute the VDF on top of the last block and the grinding can start again. Thus after at least k iterations of grinding, following standard results on Binomial distribution, the honest players contribute a block with probability

$$1 - (1 - p_h)^k = 1 - ((1 - p)^{n-n_{\mathcal{C}}})^k = 1 - (1 - p)^{k\times(n-n_{\mathcal{C}})}.$$

Since the above probability tends to 1 as $k \to \infty$, for every probability p_1, choosing k big enough ensures that after k iterations of grinding, the honest players contribute at least one block with overwhelming probability, i.e., with probability $1 - p_1$.

Above we have proved that for each round the probability of the adversary grinding more than ℓ blocks is p_0. Thus over k independent iterations of grinding at different rounds, the probability that every single iteration contains less than ℓ blocks is $(1 - p_0)^k$. Since p_0 is chosen to be close to zero, we approximate this probability to $1 - p_0 k$. We notice that the probability that the adversary contributes more than $k \times \ell$ blocks over k iterations of grinding is strictly less than $p_0 k$ (at least one of the grinding would need to have more than ℓ blocks).

For k and ℓ big enough we have the following: the honest players contribute at least one block every k iterations of the grinding (except with probability p_1) and over these k iterations the adversary contributes at most $k \times \ell$ blocks (except with

probability less than p_0^k). Combining these two probabilities, we conclude that out of every $k \times \ell + 1$ blocks, the honest players contribute at least one block (and thus the adversary at most $k \times \ell$) with probability at least:

$$(1-(1-p)^{k\times(n-n_C)})(1-p_0 k) = 1 - kp_0 - (1-p)^{k\times(n-n_C)} + kp_0(1-p)^{k\times(n-n_C)}.$$

Since $(1-p)^{k\times(n-n_C)}$ and $kp_0(1-p)^{k\times(n-n_C)}$ tend to 0 as $k \to \infty$ and p_0 can be chosen as small as possible, for every p_α we can find appropriate p_0 and k such that $p_\alpha \geq kp_0 + (1-p)^{k\times(n-n_C)} - kp_0(1-p)^{k\times(n-n_C)}$.

One example is to choose $p_0 = 1/k^2$, in which case the above inequalities becomes $p_\alpha \geq 1/k + (1-p)^{k\times(n-n_C)} - 1/k(1-p)^{k\times(n-n_C)}$ and choosing k big enough ensures this inequality holds. In that case the value of k will determine the value ℓ (other combinations of (k,ℓ) may satisfy the inequality). This proves that, for our choice of p_α, there exists a $\mu = 1 - k \times \ell$ such that the fraction of total blocks contributed by the adversary is at most $1-\mu$ except with probability p_α.

□

Numerical evaluation To illustrate the above argument, we compute the probability $\mathcal{P}_\ell$, p_0 and p_h for specific values of n, n_C and ℓ. Using $n = 15$ and $n_C = 4$, we have that $\mathcal{P}_8 \simeq 10^{-5}$ and hence $p_0 \simeq 10^{-4}$. For $k = 15$ we have $1-(1-p)^{k\times(n-n_C)} \simeq 1 - 10^{-5}$. This would translate to $p_\alpha \simeq 10^{-3}$, and at least one honest block every $15 \times 8 = 120$ rounds.

This result may sound like a very loose bound. We highlight, however, that comparable work (Ouroboros Praos [46]) gives a similarly very loose bound. Specifically, they ensure that over any period of k blocks the probability that the adversary contributed a fraction more than $1-\mu$ blocks with $\mu = 1/k$ is no more than $\exp(\ln(R) - \Omega(k))$, where R is a length of an epoch, i.e., a sequence of blocks where the random beacon is not updated as explained in Section 4.3.3. In a subsequent paper [117], the value of k considered ranges from 50 to 1000 (for example the value of k for $p_\alpha \simeq 10^{-3}$ is around 100).

As we have argued, our proof considered a worst-case adversary and the results of our simulations will be much tighter. One way to tighten the bound in our

proof would be to consider the maximum number of blocks that an adversary can contribute over k iterations of the grinding (instead of considering that at each round the adversary grinds ℓ blocks). To the best of our knowledge, there is no closed-form formula for this number.

Next, we argue for robustness. The main reason this holds is that, by following the protocol in betting on the FCR, a block gets more references and thus has a higher score than a block not following the protocol. A coalition of players can gain a small advantage by grinding through all the blocks that they can create, but when doing so they keep their chain private and thus prevent other players from referencing it. This in turn reduces the rewards associated with these blocks.

Theorem 4.5.4 (Robustness)**.** *Given a security parameter $p_\alpha \in]0,1[$, the utility function in Equation 4.1, considering a interconnectivity parameter $k \geq 3$ and $f_R < 1/3$, $f_B < 1/3$, following the protocol is an ε-(f_R, f_B)-robust equilibrium against the attacks specified in Section 4.5.1, except with probability p_α, with*

$$\varepsilon = \min\left(\frac{f_R}{1-\mu}, 1 - \frac{\mathsf{pun} \cdot \alpha_3}{c}\right)$$

where μ is a parameter that depends on p_α, α_3 is defined as:

$$\alpha_3 = 1 - [(1-p)^{n_\mathcal{C}} + n_\mathcal{C} p(1-p)^{2n_\mathcal{C}-1} + \binom{n_\mathcal{C}}{2} p^2 (1-p)^{3n_\mathcal{C}-2} +$$
$$(n_\mathcal{C} p(1-p)^{n_\mathcal{C}-1})^2 (1-p)^{n_\mathcal{C}} + \binom{n_\mathcal{C}}{3} p^3 (1-p)^{4n_\mathcal{C}-3} + \binom{n_\mathcal{C}}{2} n_\mathcal{C} p^3 (1-p)^{3n_\mathcal{C}-2} +$$
$$\binom{n_\mathcal{C}}{2} 2n_\mathcal{C} p^3 (1-p)^{4n_\mathcal{C}-3} + (n_\mathcal{C} p(1-p)^{(n_\mathcal{C}-1)})^3 (1-p)^{n_\mathcal{C}}]$$

with $n_\mathcal{C} = \lfloor |\mathcal{P}| \times f_B \rfloor$ and $p = 1/|\mathcal{P}|$.

Argument. In the following argument, we limit ourselves to the type of strategies described in Section 4.5.1. In order to get an intuition behind the argument, we first argue that following the protocol is a Nash equilibrium. Recall that the possible choices when elected leader are: (1) whether to bet or not, (2) which leaves to include, and (3) when to broadcast their blocks. We show that for each of these, if other players follow the protocol then a player is incentivized to follow the protocol. If the player is elected leader on the FCR, we want to show that betting on it

gives them a higher probability of being a winner. This is because, by definition of the FCR (Algorithm 4.4.1), betting on it means betting on the stronger chain. As argued in Claim 4.5.1, this will lead to a higher score than any other block created, and thus by the definition of the FCR it has a higher probability of being the next block chosen by the FCR. (It could still, however, lose against another bet on the FCR that has a smaller hash, but even in this case the Label function still labels it as neutral.) This establishes (1).

To create a block on top of a weaker chain, a player needs to ignore the stronger chain, which means referencing fewer blocks, i.e., ignoring blocks of which they are aware. This means that their block will have worse connectivity, and thus has a higher chance of being labelled loser and getting a punishment. This is because, by the definition of the anticone, worse connectivity means a bigger anticone, which in turns means a bigger intersection with the blue set and thus, by the definition of Label (Algorithm 4.4.2), a higher chance of being labelled loser. This establishes (2).

To argue about (3), we now show why a rational player is incentivized to reveal their block as soon as possible. By broadcasting their block as soon as they created it, their blocks can get more references (since other players follow the protocol), which again increases the probability of being a winner, and the expected accompanying reward. (A single player has nothing to grind through.)

Now, in order to show that the protocol is ε-robust, we show that a coalition of rational players that deviate from the protocol to raise their utility, can only do so by ε (resiliency). We next show that a Byzantine adversary cannot decrease the utility of honest players by more than $1/\varepsilon$ (immunity).

Resilience We consider a coalition of a fraction f_R of rational players. As a reminder for resilience we want to verify that following the protocol is an equilibrium even when a coalition is allowed to form. We thus assume that every player outside of the coalition follows the protocol and verify that, in this condition, it is also an ε-best strategy for the coalition to follow the protocol. Because it is free to cre-

ate blocks, a rational coalition can clearly gain an advantage by grinding through all the blocks they can create in order to find a subDAG that increases their utility. However due to the restrictions imposed on our adversary (see Section 4.5.1) the advantage they can gain is limited. As shown in Theorem 4.5.3, for a probability p_α, there exists a parameter μ such that the maximum fraction of blocks contributed by an adversary that adopts a grinding strategy is $1-\mu$ (except with probability p_α) versus f_R for a coalition that follows the rule. Rational participants can thus increase their gain from $f_R \times c$ to at most $(1-\mu) \times c$. We thus achieve ε-robustness with $\varepsilon = f_R/(1-\mu)$.

Immunity For immunity, we need to show that even in the case where a fraction t of players behave completely irrationally, the outcome for the rest of the players who follow the protocol stays unchanged. According to the utility functions defined in Section 4.4, there are three independent components to the utility function that an adversary could try to influence to harm altruistic players: (1) the rwd term (2) the pun term and (3) the bigpun term. To harm the honest players, an adversary could try: (1) preventing an honest player from contributing blocks to the main chain; (2)-(3) increasing the number of blocks from the honest players that get punished by pun or bigpun. The adversary cannot incur any bigpun to the altruistic players since they cannot force them to ignore their own block, thus we only focus on cases (1) and (2). To do (1) an adversary cannot indeed prevent players from creating blocks but once they produce it, they can try and create an alternative blockDAG so that the altruistic player's block does not make it to the main chain. To do (2), the adversary could create an alternative blockDAG that does not reference altruistic players' blocks to try and incur a punishment to their blocks. (Following the Label function defined in Section 4.4, a block gets a punishment if its anticone intersects the blue set for more than k blocks and a block that has fewer connections to other blocks has a bigger anticone.) In both cases the Byzantine adversary harms a player the most when creating the biggest alternative blockDAG that does not reference altruistic players' blocks.

To incur a punishment to the altruistic players Byzantine players need to create

a subDAG of more than k blocks on their own, where k is the interconnectivity parameter. Indeed if they do so then these k blocks will be in the anticone of an altruistic player that contributed a block at that same time and will be labeled a loser according to the Label algorithm in Section 4.4.1. When choosing $k = 3$, as suggested by [56], and using similar probabilities as in Theorem 4.5.3, one can compute that the probability of creating a subDAG of more than 3 blocks is $\alpha_3 :=$ $1 - \mathbb{P}$[creating a subDAG of 0,1,2 or 3 blocks].

Additionally, we have

$$\mathbb{P}[\text{creating a subDAG of 0,1,2 or 3 blocks}] = \sum_{i=0}^{3} \mathbb{P}[\text{creating a subDAG of } i \text{ blocks}].$$

We can compute each term separately as follows. For $i = 0$, following traditional results on Binomial distribution we have:

$$\mathbb{P}[\text{creating a subDAG of exactly 0 blocks}] = (1-p)^{n_C}.$$

For $i = 1$, the only subDAG possible is to have exactly one leader in the first round (which could be any of the n_C players and hence happens with probability $n_C p(1-p)^{n_C-1}$) and then zero leader elected after this (which happens with probability $(1-p)^{n_C}$). This gives:

$$\mathbb{P}[\text{creating a subDAG of exactly 1 blocks}] = n_C p(1-p)^{2n_C-1}.$$

For $i = 2$, the adversary could have one leader elected in the first round and one in the second round and then zero or two leaders elected in the first round and then zero. The former happens with probability $(n_C p(1-p)^{n_C-1})^2(1-p)^{n_C}$ and the latter with probability $\binom{n_C}{2} p^2(1-p)^{n_C-2} \times (1-p)^{2n_C}$ because after the first round, the adversary that grinds can now toss $2 \times n_C$ coins as it has two leaders in the first round. This gives us:

$$\mathbb{P}[\text{creating a subDAG of exactly 2 blocks}] = \binom{n_C}{2} p^2(1-p)^{3n_C-2} + (n_C p(1-p)^{n_C-1})^2(1-p)^{n_C}.$$

For $i = 3$, similarly we consider all the possible combinations of subDAGs with exactly three blocks. The adversary could have 3 leaders in the first round

and then zero with probability $\binom{n_C}{3}p^3(1-p)^{n_C-3} \times (1-p)^{3n_C}$. (As before, in the second round the adversary tosses $3n_C$ coins as it has three leaders in the first round.) The adversary could also have one leader in the first round, then two leaders and then zero with probability $\binom{n_C}{2}n_C p^3(1-p)^{3n_C-2}$ or two then one then zero with probability $\binom{n_C}{2}2n_C p^3(1-p)^{4n_C-3}$ and finally one then one then one and then zero with probability $(n_C p(1-p)^{(n_C-1)})^3(1-p)^{n_C}$. Putting all of this together gives the following probability

$$\mathbb{P}[\text{creating a subDAG of 0,1,2 or 3 blocks}] = (1-p)^{n_C} + n_C p(1-p)^{2n_C-1} + \binom{n_C}{2}p^2(1-p)^{3n_C-2} + (n_C p(1-p)^{n_C-1})^2(1-p)^{n_C} + \binom{n_C}{3}p^3(1-p)^{4n_C-3} + \binom{n_C}{2}n_C p^3(1-p)^{3n_C-2} + \binom{n_C}{2}2n_C p^3(1-p)^{4n_C-3} + (n_C p(1-p)^{(n_C-1)})^3(1-p)^{n_C}$$

In turn we can compute $\alpha_3 = 1 - \mathbb{P}[\text{creating a subDAG of 0,1,2 or 3 blocks}]$. So at each round the adversary has a probability α_3 of incurring a punishment to the honest players by creating an alternative subDAG and publishing it right away. Since at each round, each player is equally likely to have the winning block, the expected utility of every player at each round is $1/n \cdot c$. Hence a fraction f_B of Byzantine players will reduce the payoff of an honest player on average from $1/n \cdot c$ (expected payoff at each round without adversary) to $(c - \mathsf{pun}\alpha_3)/n$ (expected payoff when a financial punishment is incurred with probability α_3). This proves that the protocol is ε_3-immune against a coalition of f_B with $\varepsilon_3 = \frac{(c-\mathsf{pun}\alpha_3)/n}{1/n\cdot c} = 1 - \frac{\mathsf{pun}\cdot\alpha_3}{c}$.

In the case of $k > 3$, it is straightforward to argue that the protocol is ε_k-immune with $\varepsilon_k = 1 - \frac{\mathsf{pun}\cdot\alpha_k}{c}$ where $\alpha_k = \mathbb{P}[\text{creating a subDAG of more than k}]$. Since $\alpha_k < \alpha_3$, $\varepsilon_k \geq \varepsilon_3$ and hence the protocol is also ε_3-immune against a coalition of f_B.

This proves that the protocol is ε_3-immune, for every $k \geq 3$ against a coalition of f_B with $\varepsilon_3 = 1 - \frac{\mathsf{pun}\cdot\alpha_3}{c}$.

If ε_3 is smaller than the ε derived from resiliency, then we set $\varepsilon = \varepsilon_3$ in order to obtain ε-robustness. We thus obtain ε-robustness with $\varepsilon = \min(f_R/(1-\mu), 1 -$

$\frac{\mathsf{pun}\cdot\alpha_3}{c}$.

□

Finally, we argue for convergence. Intuitively, the main argument here is that if an adversary tries to grow multiple chains to prevent altruistic players from agreeing on a main chain, altruistic players are still very likely to agree on a chain, since the score of the main chain grows more than the score of other chains. Additionally, since an adversary is unlikely to be elected leader for consecutive blocks, it is unlikely that other players will revert their main chain once they have agreed on one. Even if the adversary somehow manages to build their own private chain, e.g., by using the VDF or bribing old participants in a long-range attack, the decentralized checkpointing mechanism in Fantômette provides a notion of finality for blocks. Thus, by the time they succeed in mounting such an attack, it will be too late and other participants will not accept their chain.

As explained in Section 4.2, for convergence we consider a coalition of $f_B < 1/3$ Byzantine players and $f_A = 1 - f_B$ altruistic players. For this argument, we extend the adversarial behaviour defined in Section 4.5.1 to consider (1) an aborting adversary that is trying to prevent the DAG from growing (2) an adversary that managed to privately create a fork that lasts over a period longer than the checkpointing mechanism (by, for example, mounting a long-range attack as explained in Section 2.2.3).

Theorem 4.5.5 (Convergence)**.** *Given a coalition $f_B < 1/3$ of Byzantine players defined as above and $f_A = 1 - f_B$, we have that for every $k_0 \in \mathbb{N}$, there exists a chain $\mathsf{C}^0_{k_0}$ of length k_0 and time τ_0 such that for all altruistic players i and times $t > \tau_0$: $\mathsf{C}^0_{k_0} \subseteq \mathsf{C}^{i,t}$, except with negligible probability, where $\mathsf{C}^{i,t}$ is the main chain of player i at time t.*

Argument. We start by showing that the length of the longest chain in the DAG, indeed grows. This is relatively straightforward, and follows from the discussion around liveness in Section 4.4.1. To summarize, even if a Byzantine player is the only leader and chooses not to publish a block, after some delay players will "redraw" the lottery and an altruistic player will be chosen eventually. We now calcu-

late the worst-case growth rate of the main chain, which happens when the adversary simply aborts. If an adversary controls n_C players, no honest leader is elected with probability $(1-p)^{|\mathcal{P}|-n_C}$. This means that after each block, the chain will grow normally with probability $1-(1-p)^{|\mathcal{P}|-n_C}$ and will grow with a delay of δ (where δ is the delay in the VDF) with probability $(1-p)^{|\mathcal{P}|-n_C}$. Moreover since a block is propagated with a maximum delay of Δ, we have that the worst rate at which a block is created is $(1-(1-p)^{|\mathcal{P}|-n_C})\cdot\Delta+((1-p)^{|\mathcal{P}|-n_C})\cdot(\Delta+\delta)=\Delta+\delta\cdot(1-p)^{|\mathcal{P}|-n_C}$. However it could be the case that more than one chain grows in the DAG. We now move on to prove the core of the proof.

Let k_0 be an integer. Due to the previous argument about the growth of the chain and the semi-synchrony assumption (all players receive a block before Δ slots), there exists a time τ_1 such that every honest player has in their DAG at least one chain C_{k_0} of length k_0. Let's assume that there exists two such chains C_{k_0} and C'_{k_0}. We show that, with high probability, after some time τ_0 one will be "dropped" and thus for the remaining one we will have that for every $t>\tau_0$, $C^0_{k_0}\subseteq \mathsf{C}^{i,t}$.

Let's assume that players start creating blocks on both chains. After some time less than or equal to Δ players will be aware of the other chain and thus can start referencing it (Δ is smaller than δ). Then it has to be the case that the score of one chain will grow more than the other one as shown in Claim 4.5.1 (even if both chains keep growing, they will not grow at the same rate). Now, we argue why it is unlikely that both chains keep growing indefinitely. The weaker chain grows only if the leader on that chain is either Byzantine or has not heard of the latest blocks on the other chain. For every altruistic player, for two chains of the same score created at the same time, there is a probability $1/2$ that they receive the weaker chain first due to the unbiasability and uniform distribution of the random beacon. Thus for an altruistic player there is half a chance that they extend the weaker chain. Even an adversary that would try to delay some of its blocks on purpose to increase the number of forks would not be able to choose on which candidate to delay which block due to the private unpredictability of the leader election (e.g., the adversary could not try to delay the block on the stronger chain to the leader on the weaker

chain but not the leader on the stronger chain to ensure that both are extended).

On the other hand, the stronger chain grows even if the elected leader received the weaker one first (as long as they are not eligible on it, which happens with probability $1-1/n$). More formally, in the case where altruistic players are leaders on both chains, the stronger one will be extended with probability $(1-1/n)+1/n\cdot 0.5 = 1-0.5\cdot 1/n$ and the probability that the weaker chain grows is 0.5. Thus it is more likely for an altruistic player to extend the stronger chain. This explains why the strongest chain grows with a higher probability.

Thus with high probability, there exists a time τ_0 such that altruistic players will stop extending the weaker chain.

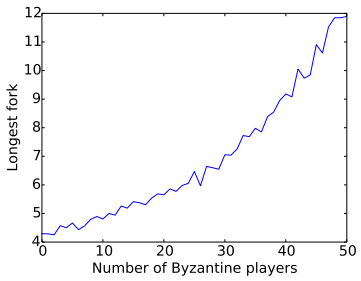
After this time τ_0 it is very unlikely that players will revert their main chain to another chain. Indeed an adversary that tries to revert the main chain does not succeed except with negligible probability. Let's assume that at time τ_0, the difference between the main chain and the chain that the adversary is trying to extend is m. To revert the chain, they need to create a competitive DAG with at least m references within the fork faster than the main chain grows. As m gets bigger, the adversary will need to create more blocks privately and this attack becomes less likely to succeed as the probability of creating ℓ blocks privately decreases with ℓ (this probability is similar to the probability in the proof of Theorem 4.5.3).

Finally, as explained in Section 2.2.3, we must consider long-range attacks, where an adversary re-writes the history by bribing old participants. Although this behaviour does not fall within our adversary specifications, we still consider it, as it is a well-known attack within PoS systems. Because we add a decentralized checkpointing, this attack will not succeed. Let's assume that an adversary has bought old keys from previous participants and re-wrote the history of the blockchain with those. When they receive this new chain, altruistic players are already aware of at least one attestor block (the reconfiguration period has to start after an attestor block as explained in Section 4.4.1). According to the betting rule in Section 4.4.1 once altruistic players know of an attestor block, they will not bet on a block that does not bet on an associated candidate block. Thus altruistic players will never bet on

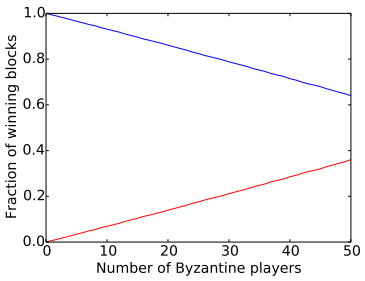
the new adversarial chain. This is also true for rational players since when they see this new chain, they would have to start ignoring all the blocks they have created in order to bet on it (due to the fourth check in VerifyBlock), thus losing most of their deposit. Thus the chain created with old keys will not be accepted by current participants.

We have argued that after time τ_0, altruistic players have agreed on the main chain C_{k_0} and that it is very unlikely they will revert to another main chain. This proves the result. □

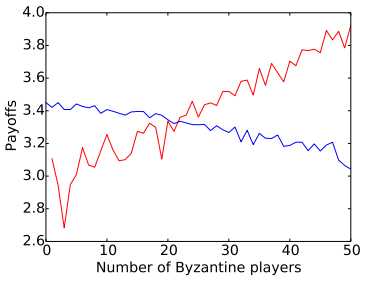
4.5.3 Simulations



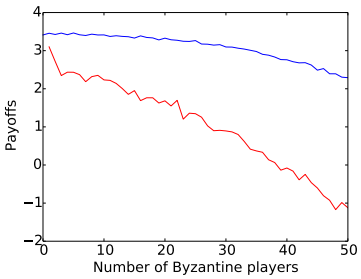
(a) The length of the longest fork, in the presence of a coalition of Byzantine players.



(b) The fraction of winning blocks belonging to altruistic (blue) and Byzantine (red) players.



(c) The payoff for altruistic players (blue) and a coalition of rational players (red).



(d) The payoff for altruistic players (blue) in the presence of a coalition of Byzantine players (red).

Figure 4.5: Results from our simulation, averaged over 150 runs and considering up to 50 non-altruistic players (out of a total of 150).

We next present our simulations that support the arguments above. As stated

in Section 4.2, we consider three types of players: Byzantine, altruistic, and rational. Byzantine and rational players are limited to the behaviours described in Section 4.5.1. We simulate the game using different fractions of different types of players. Our simulation is written in Python, and consists of roughly 1,000 lines of code. All players start with the same deposit. To model network latency, we add random delays between the propagation of blocks amongst players. Following Decker and Wattenhofer [118], this random delay follows an exponential distribution. Each simulation that we run has 150 players and lasts for 5,000 time slots. All our results are averaged over 150 runs of the simulation.

In addition to the balance of the types of players, there are several different parameters we need to consider: k, the inter-connectivity parameter; c, the constant in the reward in Equation 4.1; pun, the punishment; and bigpun, the big punishment. We chose $k = 3$, $c = 1$, $\mathsf{pun} = 6$, and $\mathsf{bigpun} = 10$. We also define the initial deposit of all players to be 0, but allow for payoffs to go below zero. We stress that all these values are relatively arbitrary, as we are more interested in the ratio between them and their evolution rather than their specific values. Different parameters do, however, result in different effects on the protocol. For example, decreasing the value of the punishment would increase the immunity of the protocol, but would also weaken its resilience (since a coalition would be able to gain a bigger profit). We leave a more in-depth exploration of this trade-off as an open problem.

Because we do not use our simulation to support our argument for the growth of the chain, we do not implement the VDF function discussed at the end of Section 4.3.1 (as it is crucial only for liveness).

Strategies As explained in Section 4.5.1, the strategy followed by rational players is to create all the blocks they can privately and then grind through all the subDAG to find the one that increases their expected utility and broadcast this subDAG only. For the Byzantine players, we are interested in capturing the *worst* type of adversary in terms of (1) decreasing the payoff of altruistic players (this will capture the immunity property) and (2) the biggest fork they can create (to capture the convergence property). We explained in the proof of Theorem 4.5.4 that the biggest harm

Byzantine players can incur to altruistic players is by creating the biggest subDAG possible. Thus we consider Byzantine adversaries that create as many blocks as possible until they are no longer elected and broadcast them.

Results We measured the length of the longest fork; the results are in Figure 4.5a. For a maximum number of 49 Byzantine players, the longest fork is 12 blocks. As in this PoS setting blocks are created instantaneously, unlike PoW, this value is acceptable, so the simulation supports our argument that the protocol converges.

In Figure 4.5b the maximum fraction of blocks contributed by a fraction of up to one-third of non-altruistic players is 0.36 compared to an expected contribution of $49/150 = 0.327$ if that coalition was honest, meaning that grinding through all the blocks they are eligible to produce allows an adversary to contribute 3.3% more blocks on average. This shows the chain quality property with $\mu = 1 - 0.36 = 0.64$. (This result is, as expected, better than what was discussed in the argument of security, which was a loose bound.)

For resilience, we see in Figure 4.5c that a coalition of up to one-third of rational players increases their payoff up to 3.8 compared to 3.5 when not forming a coalition. We thus achieve ε-resiliency with $\varepsilon = 1.086$. To quantify the harm that a Byzantine coalition can do to others, we compute for each simulation the payoff of altruistic players in the case of a fraction t of Byzantine players trying to harm the players. Figure 4.5d shows the payoffs for both altruistic and Byzantine players. The payoffs of altruistic players are unaffected even in the presence of a quarter of Byzantine players, but start decreasing after the coalition becomes larger. We thus achieve immunity for coalitions up to a quarter, but do observe here that the payoff of the Byzantine coalition is much more negatively affected than that of the altruistic players.

4.6 Limitations

Our work proposes a new consensus protocol, Fantômette, as well as new security considerations for consensus protocols but still suffers from some limitations that we highlight here.

1. We did not consider a general form of adversary and have limited our adversary to the specific strategy that we have explained in Section 4.5.1. Although this limits the generality of our work, this allowed us to keep the private unpredictability property of the random beacon. We also constructed a new type of argument that we hope can be built upon in a more general setting. Different avenues to incorporate a more general adversary could be considered; e.g., we could consider a model similar as the one presented in Ouroboros Praos [46] and lose our private unpredictability property as explained in Section 4.3.3. We leave them as open problems.

2. We presented only arguments, supported by simulations, to claim security and not formal proofs. One reason for doing so is that the field of blockchain consensus protocols is a rather new one and no proof techniques have been well established and widely adopted in this community. We hope this work can contribute to the establishment of a new framework for consensus protocols.

3. We considered only the incentives related to block rewards as we believe other costs to be negligible compared to the rewards associated with running the consensus protocol. Specifically:

    - We ignored the application layer of the protocol, more precisely we did not consider the impact of transactions (nor transaction fees) within our analysis.
    - We did not include the cost of running a VDF in our treatment of incentives.
    - We ignored the network layer, e.g., costs of propagating blocks and transactions.
    - We did not analyze the complexity of our Label algorithm.

4. In our security arguments, we did not consider the three types of players (Byzantine, altruistic and rational) simultaneously. Although we explained

that having the robustness property could be used to argue that rational players were acting altruistically.

We leave a more thorough analysis of our protocol without the limitations above as an interesting open problem and believe that, although limited, our arguments for security present a novel approach that could help build a new security model for blockchains. We also believe the novel design of our protocol brings some new interesting ideas, such as considering different types of pointers (references and bets) that other protocol designers could be inspired by.

4.7 Conclusions

We presented Fantômette, a PoS consensus protocol composed of: a leader election protocol, Caucus, a scheme for incentivization and a decentralized checkpointing mechanism. The protocol satisfies convergence and chain quality in a semi-synchronous setting (without synchronized clocks) against a coalition consisting of up to a third of participants subject to our specific adversarial model, and achieves finality via decentralized checkpointing. Additionally, Fantômette satisfies game theoretic properties (within our specified adversarial model): $(f_R, f_B) - \varepsilon$ robustness.

While Fantômette makes some important first steps in treating incentives as a first-class concern, there are other avenues to consider. In terms of evaluation, existing literature analyzing incentives in PoW-based systems has used techniques like Markov Decision Processes [64] or no-regret learning [13] in order to justify more formally the best rational strategy. These techniques would be much more difficult to apply in a setting with PoS, but it would nevertheless be useful to better justify the Byzantine strategy used in our simulations.

Finally, we currently treat all bets as being of equal value (one block is one bet), but it may be interesting to consider bets of variable size, in which players that are more confident about the blocks on which they place bets (for example, because those blocks are highly connected) could attempt to gain a higher reward by placing a bet of higher value.

Chapter 5

Egalitarian Society or Benevolent Dictatorship: The State of Cryptocurrency Governance

> *L'autorité repose d'abord sur la raison. Si tu ordonnes à ton peuple d'aller se jeter à la mer, il fera la révolution. J'ai le droit d'exiger l'obéissance parce que mes ordres sont raisonnables.*
> Antoine de Saint-Exupéry - Le Petit Prince (1943)

In the previous chapter, we constructed a decentralized consensus protocol. As we have seen, the designers of a cryptocurrency necessarily make numerous decisions regarding the security of both the high-level protocol and the implementation itself. The governance is indeed an inherent central bottleneck of any decentralized currency.

Before the cryptocurrency community even begins to address the question of designing an effective decentralized governance structure, we need a way to compare and evaluate any proposed solution. With this question in mind, this chapter aims to analyze the level of decentralization in the governance of the two most prominent cryptocurrencies [23]: Bitcoin and Ethereum. As we have highlighted in our literature review in Section 3.5, most of the work on the centralization of the governance structure of cryptocurrencies is rather theoretical. Similarly, governance structures in open-source systems have been largely studied from a social

science point of view [119, 120]. This chapter, on the other hand, proposes a quantitative analysis. There exists a large body of work on code repository mining [121, 122, 123] but they are mostly looking at the content of the repositories (e.g., finding patterns or detecting bugs) whether our work is focused on the contributors. Specifically, we are interested in the number of contributors and their level of contributions (for example in terms of the number of comments) and in ways to quantify the centralization of the governance process. For example, the ideal decentralized system would have a large number of contributors where all contributors contribute uniformly (i.e., the same amount). More realistically, no contributors should have a disproportionate amount of contributions as this would indicate a centralized trend. As it is not always meaningful to have an absolute quantification for decentralization, we use our metrics primarily to compare Bitcoin and Ethereum.

Our Contributions

In this chapter, we study the centralization in the existing governance structures of Bitcoin and Ethereum by looking at their GitHub repositories. We would like to verify that their GitHub contributions are truly decentralized in the sense that a sufficient number of people are contributing and their contributions are roughly uniformly distributed, i.e., without one, or a handful of contributors, doing the bulk of the work. In order to determine whether our results are as expected in any open source software, we also conduct our study on two popular open-source programming languages: Clojure and Rust. We measure two different properties of the data: the number of developers per file in each of the codebases; and the number of commenters in the issues and pull requests. We then compute some centrality metrics, to verify whether each contributor contributes roughly the same or whether contributions are centralized, meaning that a few percentages of all contributors do most of the work. According to these centrality metrics, Bitcoin is consistently more decentralized than Ethereum, but has a similar level of decentralization to Clojure and Rust. The median and interquartile range of contributors per file on the main code-base was 2 and 5 for Bitcoin, 1 and 1 for Ethereum, 2 and 5 for Rust and 2 and

4 for Clojure. We then verify how similar Bitcoin and Ethereum contributions are compared to Clojure and Rust. Using the bootstrap Kolmogorov-Smirnov test, we found that the number of contributors per file all came from different distributions. Furthermore, by plotting the number of main commenters on the main codebases over time, we found that for all the systems we studied, no more than 10 contributors accounted for at least 51% of the total comments, suggesting, generally, low decentralization.

To evaluate the decision-making infrastructure in Bitcoin and Ethereum, we look into who creates and comments on improvement proposals. The mean and interquartile mean of the number of comments per author was 11.4 and 6.5 for Bitcoin and 9.1 and 5 for Ethereum, indicating a centralization trend.

Finally, we compare the communities behind Bitcoin and its fork Bitcoin Cash, and Ethereum and its fork Ethereum Classic, to see whether these forks bring in new people or split the initial community. We use the Sørensen-Dice index to measure the intersection of contributors in the main code base and contributors on the improvement proposals. Intuitively a Sørensen-Dice index of 0 means that the two sets have an empty intersection and thus that the contributors of the forking cryptocurrencies were not contributing to the forked cryptocurrency while an index of 1 means the sets are identical. We got a value of 0.069 for Bitcoin and 0.108 for Ethereum meaning that the contributors in Bitcoin Cash and Ethereum Classic were mostly new. All the measurements in this chapter were made in October 2017, before Bitcoin Gold [32] and Bitcoin SV [33] existed, and come from [21].

5.1 Methodology

5.1.1 Comparison with programming languages

To determine whether the governance structures of Bitcoin and Ethereum are as decentralized as should be expected, we compare them against those of open-source, general-purpose programming languages. We chose programming languages as, similarly to cryptocurrencies, they tend to have a large amount of participation from their user communities. For an even closer comparison, we sought out programming

languages that: (1) have existed for a similar length of time to the cryptocurrency; (2) have a similar number of users (which we measured according to the number of watchers and stars on the GitHub codebase [124]); and (3) are decentralized in the sense that they are maintained by an online community rather than a private company or government. We could not find programming languages that fully satisfied each of these properties, but we decided that a relatively fair comparison was between Bitcoin and Clojure, and Ethereum and Rust.

Bitcoin and Clojure were both proposed by individuals (or a set of individuals) and were both released in 2009 (Bitcoin in January, and Clojure in May). While Bitcoin has a much larger userbase than Clojure (close to 2000 watchers and 18k stars, as opposed to roughly 700 watchers and 7k stars), we ultimately decided to stick with this comparison rather than use a programming language like Go, which does have a larger userbase, as Go is closely tied to Google.

Ethereum and Rust were both released in 2015 (Ethereum in July, and Rust in May), and are both tied to not-for-profit foundations (Ethereum with the Ethereum Foundation, and Rust with Mozilla). Rust has a larger, but not incomparable, userbase than Ethereum: roughly 1500 vs. 900 watchers, and 24k vs. 8k stars.

5.1.2 Data collection

To quantitatively measure the level of centralization in the maintenance of Bitcoin and Ethereum, we analyze their codebases and the extent to which these codebases are produced and maintained in a decentralized fashion. We obtained copies of the open-source repositories for Bitcoin, Bitcoin Cash, Ethereum, Ethereum Classic, Rust, and Clojure. A summary of the locations of these repositories is in Table 5.1.

One notable property of these platforms is that Bitcoin has only one reference client, whereas the others tend to have many. For Ethereum, we collected the repositories for all the clients as listed in the Ethereum documentation.[1] For Ethereum Classic, we considered the Go, C++, and Python clients, as the ones for JavaScript, Java, and Ruby were not listed. The Parity client supports both

[1] `http://ethdocs.org/en/latest/ethereum-clients/choosing-a-client.html#why-are-there-multiple-ethereum-clients`

Name	Repository URL
Bitcoin	`https://github.com/bitcoin/bitcoin`
Bitcoin Cash (ABC)	`https://github.com/Bitcoin-ABC/bitcoin-abc`
Clojure	`https://github.com/clojure/clojure`
Ethereum	`https://github.com/ethereum/`
Parity	`https://github.com/paritytech/parity`
Ethereum JS	`https://github.com/ethereumjs/ethereumjs-lib`
Ethereum Ruby	`https://github.com/cryptape/ruby-ethereum`
Ethereum Classic	`https://github.com/ethereumproject`
Rust	`https://github.com/rust-lang/rust`

Table 5.1: The open-source repositories for the various cryptocurrencies we consider. For Ethereum and Ethereum Classic, the listed repositories contain the code for the Go, C++, and Python versions of the client. Parity is compatible with both Ethereum and Ethereum Classic.

Ethereum and Ethereum Classic. For Bitcoin Cash, we picked the most popular one in terms of watchers and stars, which was Bitcoin ABC.

Since contribution to the protocol is also captured through discussions in addition to lines of codes written, we also scraped all the discussion threads for pull requests and issues (both open and closed). The discussions of Improvement Proposals were not included in the Bitcoin and Ethereum repositories themselves, so we also scraped the main pages, pull requests, and issues on the respective GitHub repositories for Bitcoin (BIPS) [28] and Ethereum (EIPS) [125].

5.1.3 Centrality metrics

Table 5.2 lists some of the centrality metrics we use. In addition to these, we also use the mean and the median. The interquartile range (IQR) represents where the bulk of values lie and is computed as the difference between the 75% and the 25%, and the interquartile mean (IQMean) is the mean of the data in the IQR. The benefit of using the IQMean (as compared to the regular mean) is that, as with the median, it is not affected by outliers.

Centrality metric	Usage
Interquartile range (IQR)	Measure of spread
Interquartile mean (IQMean)	Mean of the data in the IQR
Kolmogorov-Smirnov test	See if two vectors have the same probability distribution
Nakamoto index	Minimum # of contributors making 51% of the data
Satoshi index	Minimum % of contributors making 51% of the data
Sørensen-Dice index	Measure of similarity of two sets

Table 5.2: Centrality metrics used.

To confirm the statistical significance of our findings, we use a two-sample Kolmogorov-Smirnov test [126, 127], which determines whether or not two vectors of values have the same probability distribution. More specifically, it quantifies the distance between the empirical distribution functions of the two samples. The p-value, used to determine the statistical significance of the test, must be under 0.05 in order to reject the null hypothesis (i.e., in order to show that the two vectors have a different distribution). We used the Bootstrap version of the Kolmogorov-Smirnov test [128], which is designed to work on discrete distributions.

The Nakamoto index, introduced by Srinivasan and Lee [84], represents the minimum number of contributors to a dataset needed to get 51% of the data. We refer to the normalized version of this index as the Satoshi index, which represents the minimum percentage of all contributors needed to get 51% of the data. Finally, the Sørensen-Dice index [129, 130] captures the similarity of two sets. It is defined as $\mathsf{SD}(X,Y) = \frac{2|X \cap Y|}{|X|+|Y|}$, so in particular has a value of 1 for sets that are equal and 0 for sets that are disjoint.

5.2 Data Analysis

5.2.1 Contributors to the main codebase

To capture the number of people collaborating together, for each repository, we collected all non-hidden files and measured how many distinct authors had contributed to that file throughout its lifetime (in terms of Git commits). In a decentralized sys-

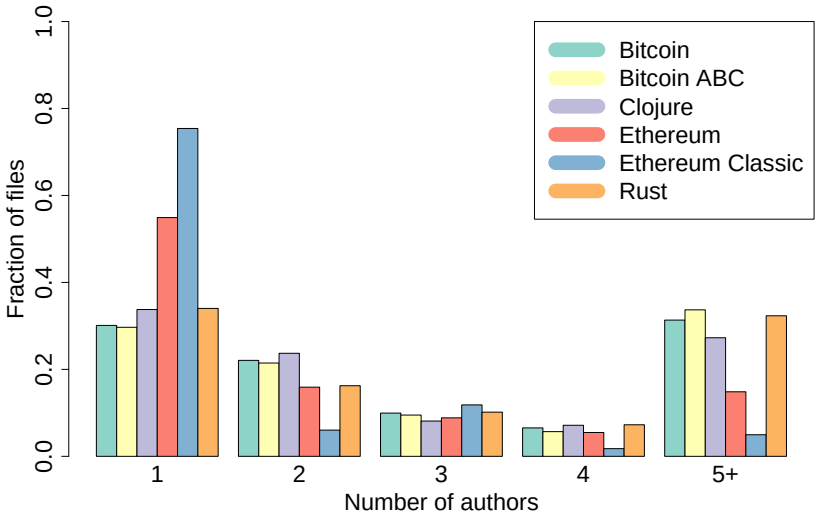


Figure 5.1: The coverage of each file in a given repository, as determined by the number of authors that have contributed to that file. Different clients are grouped according to the cryptocurrency they support.

tem we could expect to see a lot of collaboration, and thus for most of the files to have several contributors and very few files being single-authored. The results of this measurement can be seen in Figure 5.1. While the number of contributors to the Bitcoin and Bitcoin Cash codebases follows a fairly similar pattern, the number of contributors to the Ethereum Classic codebase follows a different distribution to that of the Ethereum codebase, even though Ethereum Classic is a fork of Ethereum. Both Clojure and Rust seem to follow a fairly similar pattern to that of Bitcoin.

For Bitcoin, 30% of all files were written by a single author, and 24% of these files were written by the same author, Wladimir van der Laan. This means that one author wrote 7% of the files. In Ethereum, 55% of all files were written by a single author, and 36% of these files were written by the same author, Tomasz Drwiega. This means that one author wrote 20% of the files.

Table 5.3 contains the mean, median, IQR, and IQMean for each of the repositories. We see relatively similar metrics for Bitcoin (and Bitcoin Cash) and both of the programming languages, and see that Ethereum and Ethereum Classic are both lower for all metrics. The median confirms the low level of collaboration on all these repositories.

	# Authors per file				# Comments per author			
Repository	Mean	Median	IQR	IQMean	Mean	Median	IQR	IQMean
Bitcoin	5.03	2	5	2.66	27.2	2	4	2.4
Bitcoin Cash	5.48	2	6	2.94	3.8	2	2	1.8
Ethereum	2.04	1	1	1.27	13.5	2	3	2.0
Ethereum Classic	2.27	1	2	1.78	11.1	2	3	1.8
Clojure	4.03	2	4	2.41	1.9	1	1	1.3
Rust	5.17	2	5	2.79	69.8	3	8	3.5

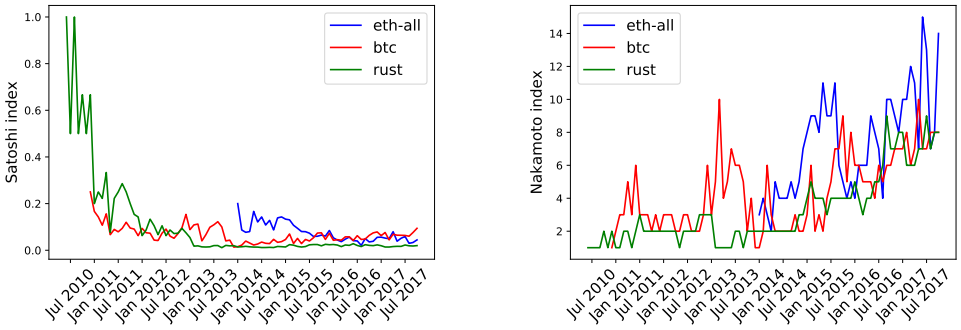
Table 5.3: Centrality metrics for the number of contributors per files in the repositories and the number of comments per author in the pull requests and issues

We perform a Kolmogorov-Smirnov test to confirm the statistical significance of our comparative findings; the resulting p-values are in Table 5.4. We see that the results are statistically significant for all the comparisons except for Bitcoin and Bitcoin Cash, indicating that the respective numbers of codebase authors are drawn from different distributions, except for Bitcoin and Bitcoin Cash. This is expected as Bitcoin Cash is a very recent fork of Bitcoin.

	Bitcoin Cash	Clojure	Ethereum	Ethereum Classic	Rust
Bitcoin	0.4749	0.001	$< 10^{-16}$	$< 10^{-16}$	0.001
Bitcoin Cash		0.003	$< 10^{-16}$	$< 10^{-16}$	0.002
Clojure			$< 10^{-16}$	$< 10^{-16}$	0.028
Ethereum				$< 10^{-16}$	$< 10^{-16}$
Ethereum Classic					$< 10^{-16}$

Table 5.4: p-values for the Kolmogorov-Smirnov test on the number of authors per file.

We acknowledge, however, that measuring levels of centralization by looking at the codebase is limited in some respects, as it is not practical — or even necessarily important for accountability — to have many people contributing to the same files, and there are likely people looking over and discussing files in ways that are not reflected in Git commits. This is why we look next at the discussion around the



(a) Satoshi index (b) Nakamoto index

Figure 5.2: The evolution of the Satoshi and Nakamoto indexes over time. The values for Ethereum are in blue, for Bitcoin in red, and for Rust in green.

code and how decentralized it looks.

5.2.2 Commenters on the main code base

To get a feeling for the evolution of the distribution of comments over the life of a cryptocurrency, we compute the Nakamoto and Satoshi indexes over time in Figure 5.2, using the discussion threads in the pull requests and issues of the GitHub repository. These graphs exclude Clojure, as ultimately we believed the dataset was too small to get any real insights. For example, there were only 72 pull requests, as compared to 11,604 for Bitcoin. Similarly, since Bitcoin Cash and Ethereum Classic are relatively recent forks, and thus have a far smaller level of discussion so far, we also exclude them from this analysis.

Intuitively, in a decentralized system we would expect the Satoshi index to be around 0.5 (i.e., if the distribution is uniform, half of the commenters should contribute, roughly, for half of the comments) and a Nakamoto index to be high (i.e., there is a large number of commenters making half of the comments as opposed to a handful). In Figure 5.2, we see that in all the repositories there is a strong tendency towards centralization in the number of commenters, with a handful of people contributing to most of the comments. The Nakamoto indexes for the codebases of Bitcoin, Rust, and Ethereum are consistently relatively low, as every month there are no more than 10 authors contributing to half of the comments for Bitcoin and Rust and 15 for Ethereum. When normalized by the total number of commenters

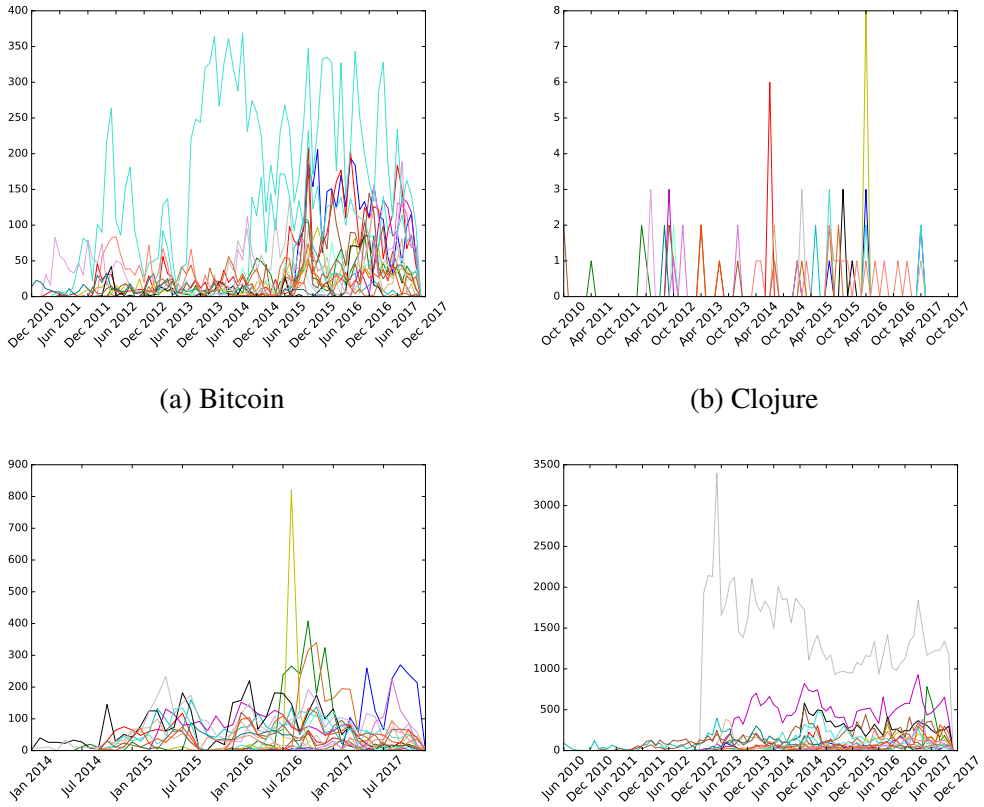
per month, for Bitcoin and Ethereum this is less than a quarter of the commenters each month (as seen in Figure 5.2a).

	10	20	30	40	50	60	70	80	90	100
Bitcoin	1	1	3	5	8	13	21	41	239	2443
Clojure	2	5	9	15	23	33	45	65	85	104
Ethereum	2	5	8	12	18	29	49	127	467	3139
Rust	1	1	1	2	4	10	21	43	181	3882

Table 5.5: Minimal number of commenters that contribute to x% of all the comments.

To see whether it was the same people making most of the comments each month or different people every time (which would indicate more decentralization), we plot in Figure 5.3 the number of comments per author every month. For Bitcoin and Rust, we see that there is one commenter that accounts for most of the comments each month (for Bitcoin, Wladimir van der Laan is the top commenter with 13,923 comments in total, followed by Jonas Schnelli with 4,409 comments), and for Ethereum there is a small handful of commenters who stand out from the rest (the top three are Gavin Wood with 3,352 total, Péter Szilágyi with 2,242, and Jeffrey Wilcke with 2,230). Overall for Bitcoin there are only eight people contributing to half of all the comments, which represents 0.3% of all commenters. For Ethereum there are 18 people (or 0.6% of all commenters), and for Rust there are four (or 0.1%). These results are summarized in Table 5.5.

This centralized trend is confirmed by the values in Table 5.3, as we see that the mean is much greater than the IQMean or median, which are values that typically ignore outliers. The mean is one order of magnitude higher than the IQMean for Bitcoin, Ethereum, and Rust. This means that the tails of the distribution (i.e., the top 25% of the distribution) differ a lot from the value in the main range. This can also be confirmed by looking at the number of comments for the top commenters, compared to the average number of comments per author. Generally, this confirms that a handful of people (less than 10) contribute to most of the comments. As this is true for all the repositories, we conclude that this is potentially a common (and



(a) Bitcoin (b) Clojure

(c) Ethereum (d) Rust

Figure 5.3: Number of comments per commenters per month.

somewhat natural) feature in open-source systems.

We also compute, in Table 5.6, a Kolmogorov-Smirnov test on the total number of comments per author. We see that the number of comments per author from Bitcoin, Ethereum, and Rust are drawn from different distributions. In the next two sections, we will focus on Bitcoin and Ethereum, looking more closely at the improvement proposals process in Section 5.2.3 and comparing the communities behind the main codebases, the improvement proposals, and forks in Section 5.2.4.

5.2.3 Improvement Proposals for Bitcoin and Ethereum

In this section, we look at the improvement proposal (IP) process. Together with pull requests, this is the main road to contributing to the design and development of the currency. For improvement proposals, just as for comments, we would expect to see a high number of proposers in a decentralized system. Similarly we would expect the number of proposal per author to be somewhat uniform, to conclude a high

	Bitcoin ABC	BIPS	Clojure	Ethereum	Ethereum Classic	EIPS	Rust
Bitcoin	0.045	0.04	0.113	0.029	0.583	0.414	$< 10^{-16}$
Bitcoin ABC		0.008	0.142	0.027	0.12	0.041	$< 10^{-16}$
BIPS			0.015	0.434	0.958	0.285	0.712
Clojure				0.033	0.043	0.07	0.021
Ethereum					0.857	0.536	$< 10^{-16}$
Ethereum Classic						0.854	0.873
EIPS							0.044

Table 5.6: p-values for the number of comments per author

level of decentralization. For each author we count how many improvement proposals they made to Ethereum and Bitcoin, and what states these proposals were in (i.e., if they were accepted, rejected, or under review). In Figure 5.4a, we notice that only a handful of people are contributing to Bitcoin improvement proposals (BIPS). In Figure 5.4b, there is mostly just one person, Vitalik Buterin, that is contributing to Ethereum improvement proposals (EIPS).

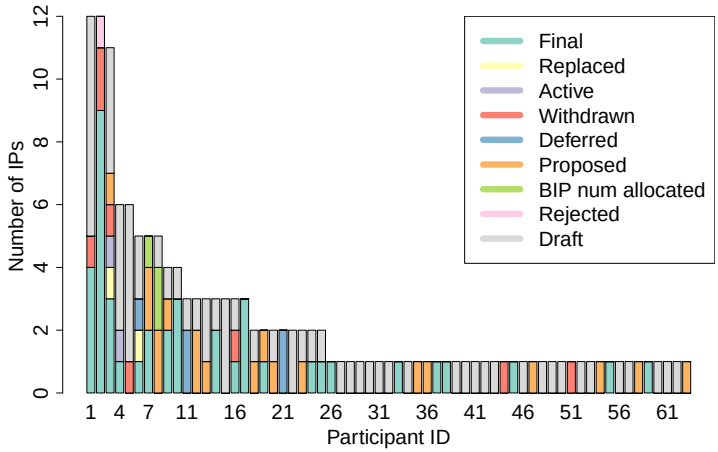
There are usually many people contributing to the discussion for every proposal, so we measure the level of centrality in terms of the number of comments in pull requests for each user in the BIPS and EIPS repositories. The results are in Table 5.7. The trend here is similar to the one observed in the previous section: the datasets contain many outliers, corresponding to the top 25% of commenters who comment significantly more than the rest. Just as before, these results do not align with our expectations of an ideal decentralized system.

5.2.4 Diversity of communities

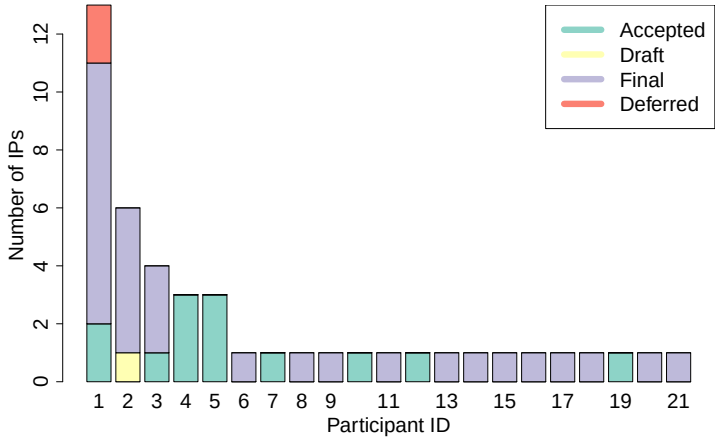
In this section, we look at whether or not the same people contribute to the discussion in the main codebase and in the improvement proposals, and whether or not there is any similarity between the community behind a cryptocurrency and its fork; i.e., any resemblance between Bitcoin and Bitcoin Cash and between Ethereum and

	Mean	Median	IQR	IQMean
BIPS	11.41	2.0	6.5	2.95
EIPS	9.16	2.0	5.0	2.56

Table 5.7: Centrality metrics for the number of comments per author.



(a) The authors of BIPs, identified by a unique numeric value, along with the number of proposals they have created and the respective status of those proposals. The top five contributors are Gavin Andresen (with 12 proposals and 9 accepted) and Pieter Wuille (12 proposals, 4 accepted), Luke Dashjr (11 proposals), Eric Lombrozo (6 proposals), and Johnson Lau (6 proposals).



(b) The authors of EIPs, identified by a unique numeric value, along with the number of proposals they have created and, if in the main set of EIPs, the status of those proposals. The top three contributors are (1) Vitalik Buterin, with 13 proposals, 11 of which were accepted or finalized; (2) Alex Beregszaszi, with 6 proposals; and (3) Nick Johnson, with 4 proposals.

Figure 5.4: Improvement Proposals for Bitcoin and Ethereum.

Ethereum Classic. Because Ethereum Classic does not have a separate implementation for every client, we focus in this section only on the Go client for each platform,

as it is the most popular. The ability to fork is one of the most important features of a decentralized system and comparing a cryptocurrency and its fork helps understand the dynamics behind the communities contributing in both.

To do this, we first compute the Sørensen-Dice index on the set of the 30 top commenters, which accounts for roughly 75% of all the comments in the relevant repositories (see Table 5.5). As we see in Table 5.8, the set of main commenters in the main Bitcoin repository and in the BIPS repository overlap, with a Sørensen-Dice index of 0.5. This means that out of the 30 main commenters of Bitcoin and BIPS, 15 are in both communities. This is much more than for Ethereum compared to EIPs, with 7 commenters in both sets.

To include all commenters, we use a weighted version of the whole set of commenters. To do so, we weight commenters by their number of comments and then compute the Sørensen-Dice index on these augmented sets. The results are in Table 5.8. Taking the weight (and all the commenters) into account, the similarity between Ethereum and EIPS is still meaningful, with an index of 0.108. The value for Bitcoin vs BIPS, however, drops to 0.069. Therefore, although half of the main commenters for Bitcoin also comment on BIPS, they do not write as many comments in the BIPS repository.

The overlap in the communities of Bitcoin and Bitcoin Cash, and Ethereum and Ethereum Classic, is small. The Sørensen-Dice index was 0.033 for both. Hence only one of the top commenters of the main repository is also a commenter in the forked one. This low value shows that the forked currency is really the formation of a new community rather than a separation of the initial one.

5.3 Discussion

According to our metrics on the number of contributors per file in the codebases, we found that Bitcoin, Rust, and Clojure were all more decentralized than Ethereum, even given the fact that Ethereum has many more reference clients. The distributions of the number of authors for all the codebases was different except for Bitcoin and Bitcoin Cash, which is not surprising given that Bitcoin Cash is a recent fork

	Sørensen-Dice Coefficient	
Repositories	Top 30	All (weighted)
Bitcoin and BIPS	0.50	0.0686
Ethereum and EIPS	0.23	0.1077
Bitcoin and Bitcoin Cash	0.03	0.0050
Ethereum and Ethereum Classic	0.03	0.0030

Table 5.8: Sørensen-Dice Coefficient for the 30 most commenters and Weighted Sørensen-Dice Coefficient for all the commenters

of Bitcoin and thus their codebases are very similar. Interestingly, while Ethereum Classic is a fork of Ethereum, the number of authors on these two codebases is still quite different. However, this fork happened a longer ago and there have been numerous changes to the Ethereum Classic codebase since the fork occurred. Our data imply that one cannot necessarily assume a natural pattern for the number of authors on an open source code base.

There was a greater number of participants in the Bitcoin improvement proposals than in those of Ethereum. Although the distribution of the number of comments on the Bitcoin codebase was different from the one on the BIPS, the distribution from Ethereum was similar to EIPS (Table 5.6). The intersection between the main commenters on Bitcoin's main codebase and the commenters on the BIPS was greater than the intersection between the commenters on Ethereum's main code base and the commenters on the EIPS. However, when considering the weighted intersection, we found the opposite applied. Generally there are very few people that account for most of the comments for Bitcoin, whereas for Ethereum this number is higher.

Finally, both the Bitcoin and Ethereum communities seem relatively unaffected by the hard forks. The number of people commenting was not significantly different before and after the forks, and there was little intersection between the people participating in the original codebases and the forked codebases. This implies that the forks did not split the communities, and that a large proportion of the community

decided to stay with the original codebases. However in our discussion we only considered Bitcoin ABC, the most popular client for Bitcoin Cash, which could limit our results. We leave as an open problem the study of all the Bitcoin Cash clients. Our data implies that there could feasibly be a natural pattern in the number of comments per author in cryptocurrencies.

5.4 Conclusions

In this chapter we compared the centralization of Bitcoin and Ethereum using their GitHub repositories and their IP GitHub repositories. In part, we looked at whether the contributions were uniformly distributed or whether there were a handful of people that disproportionately contributed compared to others.

Measuring levels of centralization by looking at the codebase or by looking at specific sources is inherently limited. While our measurements captured the number of people writing code changes and commenting on the GitHub files, they do not capture the number of people voting on whether or not changes should be accepted. We also did not capture conversations appearing in other places such as on Reddit, the main forums, or the mailing lists. We considered only two main cryptocurrencies, but there is a multitude of other ones, and it would be interesting to see whether similar patterns appear in these other cryptocurrencies, or indeed in other open-source projects in general. Also, in traditional governance structures there is typically a concern over representation: is there a variety in the demographics of the people making the rules? This is potentially a concern in cryptocurrencies as well, and it would be interesting to measure the demographics of the authors and commenters on the codebases.

We are aware of two projects that aim to tackle the centralization in governance structures of cryptocurrencies: Tezos [131], a decentralized system that incorporates governance into the consensus protocol, and Steemit [132], a decentralized social media platform in which users are incentivized to post and curate content by receiving a reward in the native cryptocurrency. However, we are not aware of any studies that analyze these solutions.

Chapter 6

Who Am I? Secure Identity Registration on Distributed Ledgers

In the previous chapters we have looked at the decentralization of blockchains at the protocol level (Chapter 4) and at the governance level (Chapter 5). We now look at another level, that of applications that can be built on top of distributed ledgers.

As a more involved example, we consider the case of governments administering pensions or benefits on a distributed ledger; the argument that has been made for doing this is that it could provide recipients with better visibility into their spending and reduce fraud [133], but such programs have come under significant scrutiny [134, 135] due to the fact that they allow the government to identify its recipients on the ledger and thus track and monitor their spending.

Our focus in this chapter is on the role that registration of identity can play in public distributed ledgers and on ways to maintain a degree of decentralization when doing so. While certain settings such as the ones described above might require a centralized registration protocol (e.g., only the government can decide whether or not a user is eligible for a pension), we also consider more informal notions of registration such as the so-called "web of trust." The web-of-trust concept has historically been used solely within the setting of certificate issuance, wherein users sign each others' PGP identity certificates to vouch for their authenticity, but has recently been discussed for the more general concept of identity in distributed ledgers. These decentralized settings are particularly appealing, as they remove the

need for a single trusted party and provide an opportunity to improve privacy and availability for users.

Our contributions

In this chapter, we propose methods for achieving registration in decentralized settings — such as the web of trust — in which multiple entities, in potentially flexible configurations, can act to validate attributes of a user's identity. We consider the registration of users' pseudonyms, unless stated otherwise.

Before presenting these methods, we give in Section 6.1 some additional background on cryptographic primitives that we use in this chapter, as well as on the web-of-trust. Then in Section 6.2 we consider both the functional and security properties that we hope to achieve. In particular, we consider how to provide *privacy* for users, so that even the registrar who sees their real-world identity and signs off on their attributes cannot subsequently link that identity to the pseudonyms that the user goes on to adopt within the ledger.

We begin with a registration protocol in the style of the web of trust (but again, leveraging some of the key properties of distributed ledgers), and then build off of it to achieve protocols that provide better privacy and overall security.

Finally, in Section 6.4, we present an implementation of a decentralized registration protocol — that most closely resembles the web of trust, but allows for the *blinding* of attributes — as an Ethereum smart contract. In this setting, users can publish certain attributes (e.g., their Twitter handle) associated with their Ethereum address. Other users or institutions can then publish a signature on these attributes, reflecting a certain belief in its veracity. For attributes that the user may not want to directly link to their real-world identity (e.g., a particular Bitcoin or other cryptocurrency address), we provide a blind signing protocol in which users can publish blinded attributes on the blockchain and other users can sign them (and then the user can unblind them locally).

This chapter is based on [15].

6.1 Background

In this section, we give some additional background on cryptographic primitives used in the rest of this chapter.

6.1.1 Public-key encryption

Some of our decentralized registration protocols use public-key encryption; here we denote the appropriate generic algorithms as $c \xleftarrow{\$} \mathsf{Enc}(pk, m)$ (for encryption) and $m \leftarrow \mathsf{Dec}(sk, c)$ (for decryption). In order to maintain compatibility with Bitcoin and Ethereum, the Elliptic Curve Integrated Encryption Scheme (ECIES) provides an encryption scheme that is compatible with ECDSA; i.e., one that allows for the encryption of ECDSA secret keys.

6.1.2 Blind signatures

As initially defined by Chaum [136], a blind signature provides an interaction — denoted $\mathsf{U}(pk, m) \leftrightarrow \mathsf{S}(sk)$ — wherein a user U obtains a signature from a signer S on a message without the signer learning anything about the message. One commonly used construction is the RSA blind signature [137], which we use in our constructions due to the lack — to the best of our knowledge — of any provably secure blind signatures that are compatible with ECDSA.

Very quickly, RSA blind signatures work as follows. First, the signer generates an RSA keypair, which consists of a secret key d and a public key (e, N). To obtain a signature on a message m, the user generates a random value $r \xleftarrow{\$} \mathbb{Z}_N^*$ and computes $m' \leftarrow H(m)r^e \bmod N$, where H is a hash function. The user then sends this value m' to the signer, who computes $\sigma' \leftarrow (m')^d$. To unblind this, the user computes $\sigma \leftarrow \frac{\sigma'}{r} = H(m)^d$, which is a valid RSA signature on m.

6.1.3 The web of trust

The web of trust is a public-key authentication system established by PGP. In this setting, if Alice trusts that a certain key belongs to Bob (e.g., they have met in person), she can demonstrate this by signing his public key. The more signatures associated with Bob's public key, the more confident another user can be that this public key does indeed belong to him and not to someone who wants to impersonate

him in order to intercept his communications.

In this system, one must of course be careful that it achieves some notion of Sybil resistance; i.e., that an adversary has not simply created alternate identities in order to vouch for their own impersonated key. To do this, users in the web of trust can form a *trust path*. For example, if Alice trusts Bob's public key, and Bob trusts Dave's public key, then there is a trust path from Alice to Dave and she can have added confidence in Dave's public key (as Bob's public key, which she trusts, was used to sign it). The shorter the trust path, the stronger the trust can be in the associated public keys.

The web of trust model can then be thought of as a directed graph whose vertices are users' PGP keys and the directed edges are signatures on keys. The distance between two vertices is then the length of the shortest trust path between the two users they represent. In order to make the system Sybil resistant one could then consider having some weighting scheme such that Alice would only trust Bob if Bob's weight is higher than some threshold t.

6.2 Definitions and Threat Model

We consider a setting in which *users* maintain *attributes* about themselves and require *registrars* to vouch for these attributes. For example, in order to register the attribute "over 18 years old," a user reveals their identity to the government, who verifies their age. If they are over 18, the government registers the user's pseudonym, and they are now able to use it directly on the blockchain. For Bitcoin, we consider only the registration of pseudonyms, but in Ethereum, we consider the registration of more general types of attributes. Confirmation that the user possesses a given pseudonym may in turn be carried out by *verifiers* in order for the user to gain access to a particular service; i.e. for the users to interact with the service using their registered pseudonyms (e.g., use it to receive a pension from the government). We break this system down into four phases: (1) *setup*, in which various actors may initialize certain information about themselves (e.g., keys); (2) *registration*, in which the user interacts with the registrar(s) to register their pseudonym(s) and receive

some evidence of this; (3) *verification*, in which the user interacts with the verifier to convince them that certain pseudonyms have been registered; and (4) *revocation*, in which either the registrar or (in some cases) the user revokes the registration of their pseudonym.

In order for the system to function, we must have a way for verifiers to check certain information about users without the intervention of the registrar. Let's assume, for example, that the user wants to register as an attribute the fact that they are over 18 years old so they can use a gambling service. If the registrar must intervene in order to confirm this attribute — as in the recently proposed brokered identification systems proposed in the US and UK [138, 139, 140] — then the registrar must be online at all times and can link the user's identity with their usage of certain services, neither of which is desirable. If instead this information is stored on a blockchain, then the verification step can happen in a non-interactive, or *passive*, fashion, as the verifier can simply check for themselves if the user's pseudonym has been registered or not. If evidence of the registration is not stored on the ledger, or if additional information is needed to "unlock" it (e.g., it is encrypted), then it may be necessary for the user to send additional information to — or otherwise interact with — the verifier. We capture these two functional properties as follows:

Definition 6.2.1 (Passive/active verification)**.** *The verification process is* passive *if any verifier with access to the shared ledger can determine whether or not a given user has registered a particular attribute. The verification process is instead* active *if verifiers require additional information beyond what is available on the shared ledger.*

In order for the system to be secure, we would like to ensure that users are able to register only accurate attributes about themselves; e.g., they can register only for services, such as a pension scheme, that they are eligible to use. We must also ensure that the individual identities of users are protected and cannot be impersonated by anyone else. Once the user has completed the registration process and is interacting within the system using only their registered pseudonyms (e.g., their Bitcoin address), we should be able to ensure *privacy*; i.e., that the registrar cannot link the

user's real-world and "on-chain" identifiers (even across separate attributes). We consider the different types of security we would like to achieve as follows:

Definition 6.2.2 (Attribute integrity)**.** Attribute integrity *holds if attributes are registered only to those users to whom they belong; i.e., in the presence of an honest registrar, malicious users are unable to either register a fake attribute or one that otherwise does not belong to them, and malicious registrars are unable to impersonate an individual honest user.*

Definition 6.2.3 (Availability)**.** Availability *holds if malicious registrars are unable to deny an individual user the right to register an attribute that they possess.*

Definition 6.2.4 (Attribute privacy)**.** Attribute privacy *holds if malicious entities (i.e., registrars and verifiers who are allowed to collude) are unable to link the attributes a user claims within the system to their identity. In particular, after the registration process is complete, malicious registrars are unable to distinguish the behavior of two users within the system that have different real-world identities but the same set of attributes.*

As we will see in our constructions, while revocation is useful and often necessary — as keys are frequently compromised or lost — it also tends to require active verification, as registrations cannot be deleted from the ledger (because it is immutable) and it is difficult to efficiently prove the absence of a revocation entry while maintaining privacy. To thus separate out these complexities, we analyze our protocols separately in the cases where revocation is and isn't supported.

6.3 Decentralized Registration

In any centralized setup, we cannot satisfy availability, as the centrality of the registrar always allows it to deny users access to the system. To thus provide this property, and open the process up to more ad-hoc forms of registration, we consider a decentralized setup in which users interact with multiple registrars, and may even pick these registrars themselves.

For example, the "web of trust" reputation system can be considered a decentralized registration process in which any user can act as a registrar. The more

	Verification		Attribute integrity	Privacy
	passive	active		
Basic web of trust	●		◐	
Blinded web of trust (with revocation)		●	◐	◐
Blinded web of trust (without)	●		◐	●
Multi-Casascius	●		●	●
Mix-network	●		●	●

Table 6.1: The different properties of a blockchain-based registration protocol and whether or not they are satisfied by our various constructions. No circle indicates that the property is not satisfied, a filled circle indicates it is, and a partially filled circle indicates it is partially satisfied.

signatures one accumulates for a particular attribute, the more *trusted* that attribute can be considered. In the PGP web of trust, however, the system still uses a central website to provide the lookup and signing services. In our constructions below, we use the blockchain to provide these two services. We also consider additional decentralized protocols that provide more robust properties or are useful in settings outside of the web of trust. In a similar way, trust in one's identity can be reinforced by a diverse set of authorities — e.g., universities and governments — who might issue different types of official documents that a user can then maintain in a single portfolio on the ledger. This type of system has been proposed recently using an approach analogous to the issuance of digital certificates [93].

6.3.1 Basic web of trust

One simple way of translating the web of trust into the setting of blockchains is to have users create transactions that vouch for each others' attributes. This can be done either individually or — if a user knows in advance which other users will vouch for their attribute — as a multi-input transaction.

6.3.1.1 Construction

In the setup phase, the user optionally chooses a set of peers to validate their attribute and act as registrars. We assume each registrar creates and publishes an on-chain identity $\mathsf{addr}_{\mathsf{R}}$.

In the registration phase, the user sends their identity id and address $\mathsf{addr}_{\mathsf{id}}$ to each registrar, who determines if the address belongs to id (or just if id is a valid identity), using some off-chain mechanisms that we omit here. If it does, each registrar R_i creates a revocation keypair $(pk_{\mathsf{rev}}^{(i)}, sk_{\mathsf{rev}}^{(i)}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$, and publishes to the blockchain a transaction $\mathsf{tx}(\mathsf{addr}(\mathsf{R}_i) \to \{\mathsf{addr}_{\mathsf{id}}, \mathsf{addr}(pk_{\mathsf{rev}}^{(i)})\})$. In a basic system like Bitcoin this could involve sending a specific amount of bitcoins to both the attribute and revocation addresses, while in a more sophisticated system like Ethereum it could be a registration smart contract. Alternatively, if the set of registrars is fixed ahead of time, a user can create an n-input $n+1$-output transaction $\mathsf{tx}(\mathsf{addr}(\mathsf{R}_1), \ldots, \mathsf{addr}(\mathsf{R}_n) \to \mathsf{addr}_{\mathsf{id}}, \mathsf{addr}(pk_{\mathsf{rev}}^{(1)}), \ldots, \mathsf{addr}(pk_{\mathsf{rev}}^{(n)}))$ and, after collecting signatures on it from each registrar, publish it to the blockchain.

In the verification phase, when the user wishes to prove that they have registered the pseudonym, the verifier checks for the existence of these transactions in the blockchain, and that the output address $\mathsf{addr}(pk_{\mathsf{rev}})$ has not spent its contents. (While this may seem inefficient, if we associate with the ledger a list of unspent transaction outputs, or UTXOs, then it becomes significantly faster.)

Our approach to revocation here and in what follows is inspired by the approach of the MIT Digital Certificates project [93]. In the revocation phase, a registrar R_i can revoke their registration by spending the contents of $\mathsf{addr}(pk_{\mathsf{rev}}^{(i)})$.

6.3.1.2 Security analysis

Verification is passive, as the verifier needs to check only whether or not certain transactions are in the blockchain.

Attribute integrity is partially satisfied: restricting ourselves to the setting of on-chain pseudonyms, no registrar is able to impersonate the user, as they don't know the private key corresponding to a user's $\mathsf{addr}_{\mathsf{id}}$. We could strengthen integrity by requiring the user to also send a signature to prove its ownership of $\mathsf{addr}_{\mathsf{id}}$. Be-

cause the user can pick its own set of registrars, however, we cannot unilaterally guarantee that a user cannot register a fake attribute, as a malicious coalition of users could act to register each other's fake identities or attributes. This is the same problem faced in the web of trust, however, and it can be mitigated by having the verifier place trust only in registrars with whom they can create a trust path of a certain (short) length (see Section 6.1.3). If malicious registrars can place themselves along this trust path with a certain proximity to the verifier, this is analogous to launching a Sybil attack, which can be prevented or detected in a variety of ways [141]. Thus, if the verifier sets a low threshold for the required length of the trust path and a high threshold for the number of registrars required to have registered the attribute, we can argue that the probability that malicious users can register fake attributes is low.

Privacy is not satisfied, as every registrar sees both id and $\mathsf{addr}_{\mathsf{id}}$ at the same time.

6.3.2 Blinded web of trust

One of the problems of the previous method is that it provides no privacy. To achieve this, we provide a blinded version of the web of trust in which the user collects blind signatures from a set of nodes and the verifier then verifies the unblinded signatures. In Section 6.4, we present the results of an implementation and deployment of this approach on Ethereum.

6.3.2.1 Construction

In the setup phase, each registrar maintains as before a public on-chain identity $\mathsf{addr}_{\mathsf{R}}$ linked to a public signing key pk_{R}.

In the registration phase, the user sends their identity id to a registrar, who determines whether or not they believe the user is eligible for the service. If they do, the user and registrar engage in the blind signing protocol $\mathsf{U}(\mathsf{addr}_{\mathsf{R}}, pk) \leftrightarrow \mathsf{R}(sk_{\mathsf{R}})$ at the end of which the user obtains a signature σ such that $\mathsf{Sig.Verify}(pk_{\mathsf{R}}, pk, \sigma) = 1$ and the registrar learns nothing about pk. The registrar also creates a revocation key pair $(pk_{\mathsf{rev}}, sk_{\mathsf{rev}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^{\lambda})$, sends it to the user, publishes to the blockchain a transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}} \rightarrow \mathsf{addr}(pk_{\mathsf{rev}}))$, and maintains the mapping from

id to pk_{rev}. The user repeats this process with every registrar.

In the verification phase, the verifier verifies the unblinded signatures, and the user proves they control the revocation address pk_{rev} by signing a message using sk_{rev}. The verifier verifies this signature, checks the existence of the revocation transaction in the blockchain, and checks that $\mathsf{addr}(pk_{\mathsf{rev}})$ has not yet spent its contents.

In the revocation phase, the registrar spends the contents of $\mathsf{addr}(pk_{\mathsf{rev}})$.

6.3.2.2 Security analysis

Verification is active, as the user must provide the signatures to the verifier. To allow for passive verification without revocation, the user could, after some delay, send pk and σ back to the registrar. The registrar would then check its own signature and, if it verifies, publish to the ledger a transaction of the form $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}} \rightarrow \mathsf{addr}(pk))$. The verifier would, in this case, simply check for the transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}} \rightarrow \mathsf{addr}(pk))$ in the blockchain to verify the registration, making it passive.

If we require revocation, however, we cannot achieve passive verification. The user would need to prove to the verifier that the coins in their revocation address are unspent, but to do that they would still need to prove that they know the secret key associated with their revocation address — as otherwise they could find and use any revocation address in the ledger — which requires active participation.

Attribute integrity is partially satisfied, as malicious registrars cannot impersonate users since they do not know the private key associated with the public key they register. While the unforgeability of the blind signature guarantees that a malicious user cannot fake the approval of an honest registrar, we cannot guarantee that malicious users and registrars cannot collude to register fake attributes. Instead, we can diminish the probability of this by requiring short trust paths and numerous registrars.

Privacy is satisfied, as the unlinkability of the blind signature means that malicious registrars are unable to link id and pk. If we consider malicious verifiers as well,

however, then the verifier could collude with the registrar and use pk_{rev} to de-anonymize the user. If we ignore revocation then privacy is (fully) satisfied.

6.3.3 Multi-Casascius

In this setting, we assume that the registrar consists of several entities (e.g., different certificate authorities) that are assumed to have some level of trust in each other; in particular, one registrar must be trusted by the others to correctly verify the identity of the user. As an improvement over the previous construction, these multiple entities make the registration process anonymous and provide passive verification, even in the case where revocation is necessary.

Our solution is based on the two-factor key generation protocol used to generate physical Casascius coins [16]. In this process, the manufacturer (Casascius) encodes on a physical coin a public key and a share of the associated secret key. (Traditional Casascius coins have the full secret key, meaning the manufacturer knows it and is able to spend the contents in the same way as the person who bought it.) The user who purchases the product can then fold in their own share of the secret key (which has been communicated to Casascius in the obfuscated form of an "intermediate code"), which yields the full secret key needed to spend the coins stored in the public key; thus, only the user and not the manufacturer can spend the coins. Our solution attempts to retain this property, which allows for attribute integrity, but provides a decentralized version for use in a wider variety of settings.

6.3.3.1 Construction

In the setup phase, each registrar R_i establishes some on-chain identity $\mathsf{addr}_{\mathsf{R}_i}$ associated with a public key pk_i, and the user creates a keypair $(pk_{pub}, sk_{pub}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$. The user chooses a set of registrars with whom they want to register, as well as the order in which the registrars will proceed. (This can be thought of as either a property of the system, or as a choice made by the user that they communicate to the registrars.)

The registration phase proceeds in two phases, which are depicted in Figure 6.1. First, the user sends pk_{pub} and their real world identity id to R_1. This

registrar verifies that the user is legitimate; if so, it picks a random secret key sk_1, sends $pk_1 \leftarrow (pk_{pub})^{sk_1}$ to R_2, and keeps (for use in the second phase) the mapping $sk_1 \mapsto pk_{pub}$. Now, for all i, $2 \leq i < n$, registrar R_i picks a random secret key sk_i, sends $pk_i \leftarrow (pk_{i-1})^{sk_i}$ to registrar R_{i+1}, and keeps the mapping $sk_i \mapsto pk_{i-1}$. Upon receiving pk_{n-1}, registrar R_n also picks a random secret key sk_n and forms $pk_n \leftarrow (pk_{n-1})^{sk_n}$. It then creates a revocation keypair $(pk_{\mathsf{rev}}, sk_{\mathsf{rev}}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$ and publishes a transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}_n} \rightarrow \{\mathsf{addr}(pk_n), \mathsf{addr}(pk_{\mathsf{rev}})\})$ that acts as a registration.

In the second phase, the registrars create an *onion* [142] to send the secret keys back to the user. In particular, R_n encrypts sk_n using pk_{n-1} and sends $c_n \xleftarrow{\$} \mathsf{Enc}(pk_{n-1}, (sk_n, \perp))$ to R_{n-1}. Now, for all i, $n > i \geq 2$, R_i folds their own secret key into the onion by sending $c_i \xleftarrow{\$} \mathsf{Enc}(pk_{i-1}, (sk_i, c_{i+1}))$ to R_{i-1}. At the end, R_1 creates $c_1 \xleftarrow{\$} \mathsf{Enc}(pk_{pub}, (sk_1, c_2))$ and sends this to the user. The user can now recover all the individual sk_i values by computing $(sk_i, c_{i+1}) \leftarrow \mathsf{Dec}(sk_{i-1}, c_i)$ for all i, $1 \leq i \leq n$ (where $sk_0 = sk_{pub}$), and can thus reconstruct the public key pk_n as

$$pk_n \leftarrow (pk_{pub})^{\prod_{i=1}^{n} sk_i},$$

and the private key as $sk_n \leftarrow sk_{pub} \cdot \prod_{i=1}^{n} sk_i$.

In the verification phase, the verifier checks for the existence of the transaction $\mathsf{tx}(\mathsf{addr}_{\mathsf{R}_n} \rightarrow \{\mathsf{addr}(pk_n), \mathsf{addr}(pk_{\mathsf{rev}})\})$ in the blockchain, and verifies that the contents of pk_{rev} are unspent.

In the revocation phase, R_1 is the only registrar that can initiate revocation (as it is the only one that knows id), but R_n is the only registrar that can spend the contents of $\mathsf{addr}(pk_{\mathsf{rev}})$. Thus, R_1 starts by sending a revocation request for key pk_1 to R_2. In turn, using the mapping $sk_i \mapsto pk_{i+1}$, R_i sends a revocation request for key pk_i to R_{i+1} for all i, $2 \leq i < n$. When the request reaches R_n, they can revoke the registration by spending the coins in $\mathsf{addr}(pk_{\mathsf{rev}})$.

The previous protocol is summarized in Figure 6.1.

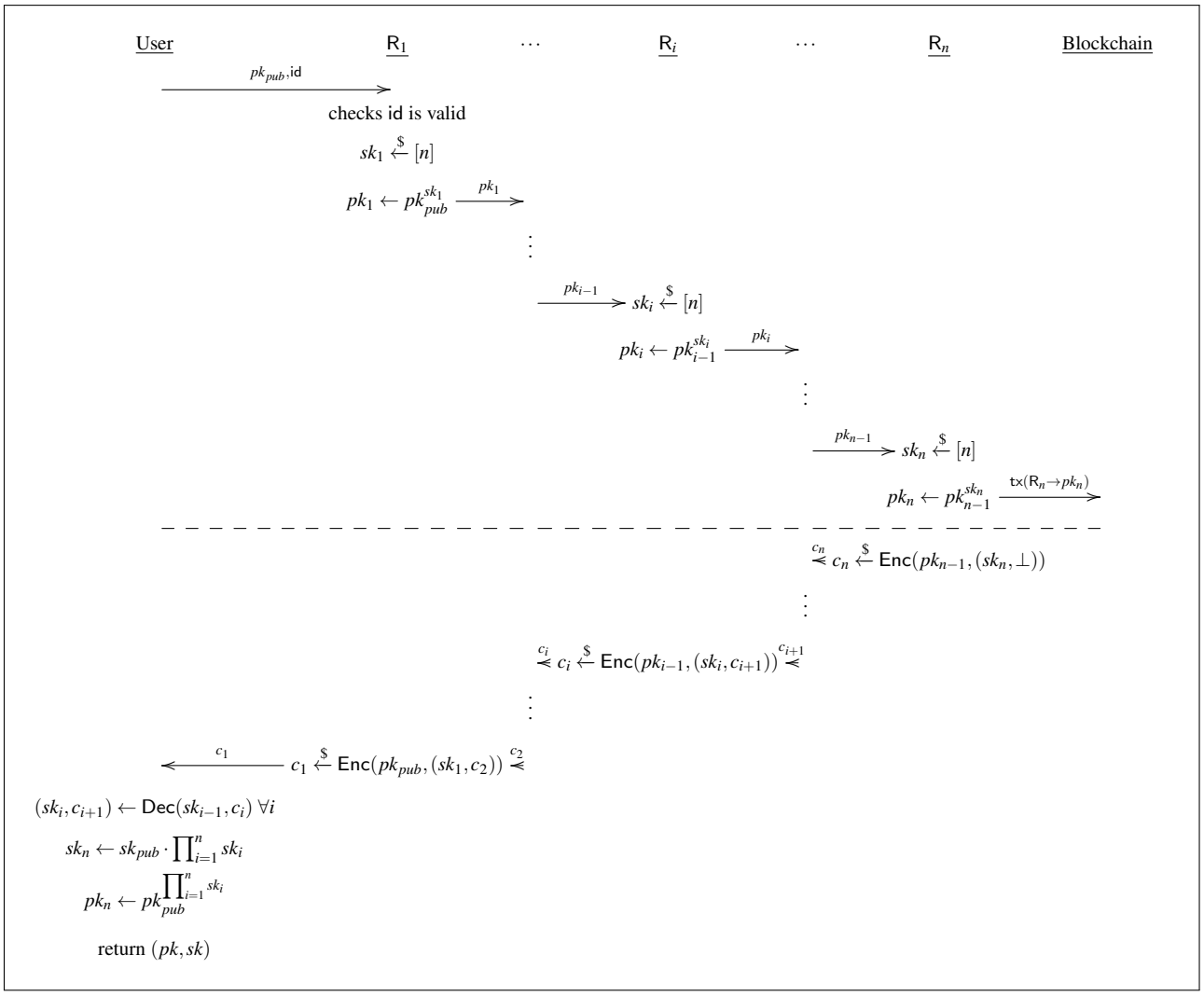


Figure 6.1: The Multi-Casacius protocol from Section 6.3.3; for ease of exposition we present a version of the protocol that does not support revocation. The dashed line denotes the separation between the first phase (the formation of the key) and the second phase (the formation of the onion) of the protocol.

6.3.3.2 Security analysis

Verification is passive, as the verifier needs to check only whether or not certain transactions exist in the blockchain.

Attribute integrity is satisfied. The first registrar R_1 checks for the validity of id, so a user cannot register a fake or ineligible identity as long as R_1 is honest.

Similarly, because the user sends a value pk_{pub} to the registrar that involves a partial secret key sk_{pub} known only to them, even if all the registrars collude the user is still the only entity who knows the full secret key associated with pk_n, which means they cannot be impersonated. As in Section 6.3.1, We could require the user to send a signature under sk_{pub} to additionally prove their ownership of pk_{pub}.

Privacy is satisfied, as long as at least one registrar is honest. For all i, $2 \leq i \leq n$, each registrar R_i knows the mapping between pk_{i-1} and pk_i, and R_1 knows the mapping between id and pk_1. If all the registrars collude, they can thus learn the mapping between id and pk_n, but as long as one registrar doesn't collude with the others and $n \geq 3$, the user cannot be de-anonymized. Assuming that R_1 does not know which node is acting as R_n, which is plausible as it communicates directly only with R_2, R_1 cannot de-anonymize the user by observing the transactions published in the blockchain. Timing attacks can be mitigated by adding some random delays in the publication of the registration transaction. Our next protocol will completely thwart this attack.

6.3.4 Mix-network

While the blinded web of trust protocol in Section 6.3.2 and the multi-Casascius protocol in Section 6.3.3 provide strong privacy guarantees, the former has the drawback that verification requires active participation on behalf of the user, and the latter has the drawback that all registrars must trust the initial one to verify the identities and allows timing attacks. Here, we try to maintain the advantages of these protocols but eliminate these drawbacks.

Without adopting the time delay from our protocol in Section 6.3.2, we cannot achieve passive verification unilaterally. Instead, we consider how to provide passive verification in a setting in which multiple users register at the same time through the same set of nodes (e.g., voter registration), which also allows us to provide each registrar with the ability to verify the set of identities for themselves without violating privacy. As we will see, if k users register at the same time then

this provides each user with an anonymity set of size k.

6.3.4.1 Construction

In the setup phase, each user j creates a keypair $(pk_{pub}^{(j)}, sk_{pub}^{(j)}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$, and each registrar R_i maintains some on-chain identity $\mathsf{addr}_{\mathsf{R}_i}$. The order of registrars is determined beforehand.

The registration phase is similar to the two-phase process in Section 6.3.3. First, each user j sends its public key $pk_{pub}^{(j)}$ and $\mathsf{id}^{(j)}$ to R_1. This first registrar then verifies that all the identities are legitimate; if not it drops the illegitimate identities and waits to receive a legitimate set of k users. For each user, R_1 then picks a random secret key $sk_1^{(j)}$, computes $pk_1^{(j)} \leftarrow (pk_{pub}^{(j)})^{sk_1^{(j)}}$, and keeps the mapping $sk_1^{(j)} \mapsto pk_{pub}^{(j)}$. It then performs a permutation π_1 on the identities and sends the public keys $\{pk_1^{(j)}\}_{j=1}^k$ and the permuted identities $\pi_1(\{\mathsf{id}^{(j)}\}_{j=1}^k)$ to R_2. For all i, $2 \leq i < n$, R_i verifies for itself the set of identities and, if they are eligible, picks for each user j a random secret key $sk_i^{(j)}$, computes

$$pk_i^{(j)} \leftarrow (pk_{i-1}^{(j)})^{sk_i^{(j)}},$$

and keeps the mapping $sk_i^{(j)} \mapsto pk_{i-1}^{(j)}$. It then applies its own permutation π_i to the mapping and sends the public keys $\{pk_i^{(j)}\}_{j=1}^k$ and the permuted identities $\pi_i \circ \cdots \pi_1(\{\mathsf{id}^{(j)}\}_{j=1}^k)$ to R_{i+1}. Finally, R_n creates k revocation keypairs $(pk_{\mathsf{rev}}^{(j)}, sk_{\mathsf{rev}}^{(j)}) \xleftarrow{\$} \mathsf{Sig.KeyGen}(1^\lambda)$ and k transactions

$$\mathsf{tx}(\{\mathsf{addr}_{\mathsf{R}_1}, \ldots, \mathsf{addr}_{\mathsf{R}_n}\} \rightarrow \{\mathsf{addr}(pk_n^{(j)}), \mathsf{addr}(pk_{\mathsf{rev}}^{(j)})\}). \tag{6.1}$$

It signs each transaction $\mathsf{tx}^{(j)}$ of this form with its private key.

In the second phase, the registrars must now jointly create the transactions to publish to the blockchain, and create an onion (as in Section 6.3.3) to send the keys back to the users. So, R_n first signs each transaction $\mathsf{tx}^{(j)}$ of the form specified in Equation 6.1 with its private key. It then encrypts $sk_n^{(j)}$ with $pk_{n-1}^{(j)}$ to form $c_n^{(j)} \xleftarrow{\$} \mathsf{Enc}(pk_{n-1}^{(j)}, (sk_n^{(j)}, \perp))$ and sends the set $\{\mathsf{tx}^{(j)}, c_n^{(j)}\}_{j=1}^k$ to R_{n-1}.

For all i, $n > i \geq 2$, R_i incorporates its own signature into the transactions $tx^{(j)}$, encrypts $sk_i^{(j)}$ with $pk_{i-1}^{(j)}$ to form $c_i^{(j)} \xleftarrow{\$} \mathsf{Enc}(pk_{i-1}^{(j)}, (sk_i^{(j)}, c_{i+1}))$, and sends $\{\mathsf{tx}^{(j)}, c_i^{(j)}\}_{j=1}^k$ to R_{i-1}.

Finally, R_1 incorporates its own signature into the transactions $\mathsf{tx}^{(j)}$ and, now that they have the full set of signatures needed for validity, publishes these transactions to the blockchain. It also creates $c_1^{(j)} \xleftarrow{\$} \mathsf{Enc}(pk_{pub}^{(j)}, (sk_1^{(j)}, c_2^{(j)}))$ and sends $c_1^{(j)}$ to each user j.

At the end, user j recovers the secret key shares $sk_i^{(j)}$ in the same manner as in in Section 6.3.3; i.e., they compute $(sk_i^{(j)}, c_{i+1}^{(j)}) \leftarrow \mathsf{Dec}(sk_{i-1}^{(j)}, c_i^{(j)})$ for all i, $1 \le i \le n$ (using $sk_0 = sk_{pub}$), and computes the secret key as $sk_n^{(j)} \leftarrow sk_{pub}^{(j)} \cdot \prod_{i=1}^{n} sk_i^{(j)}$ and the public key as

$$pk_n^{(j)} \leftarrow (pk_{pub}^{(j)})^{\prod_{i=1}^{n} sk_i^{(j)}}.$$

In the verification phase, the verifier checks for the existence of the transaction in the blockchain and verifies that the contents of $pk_{\mathsf{rev}}^{(j)}$ are unspent.

As in Section 6.3.3, the revocation request can be initiated only by R_1, but revocation can be carried out only by R_n. R_1 can initiate the process by sending a revocation request for $pk_1^{(j)}$ (which represents $\mathsf{id}^{(j)}$) to R_2. In turn, R_i transmits the revocation request to R_{i+1} using their partial key $pk_i^{(j)}$ and their knowledge of the mapping $pk_{i-1}^{(j)} \mapsto pk_i^{(j)}$. Once this reaches R_n, it can spend the coins in $pk_{\mathsf{rev}}^{(j)}$ to revoke the registration.

6.3.4.2 Security analysis

Verification is passive, as the verifier needs to check only whether or not certain transactions exist in the blockchain.

Attribute integrity is satisfied, as long as one registrar is honest: every registrar verifies the set of identities $\{\mathsf{id}^{(j)}\}_{j=1}^{k}$ for themselves, so if one registrar is honest then it will drop any fake identities and users cannot register fake ones. As in our previous protocols, malicious registrars cannot impersonate a user as they do not have access to the private key.

Privacy is satisfied, as k-anonymity is provided as long as one registrar is honest. In particular, R_i knows only the mapping between $pk_{i-1}^{(j)}$ and $pk_i^{(j)}$, and only R_1 knows the mapping between $\mathsf{id}^{(j)}$ and $pk_1^{(j)}$. Thus, as long as not all registrars collude, $\mathsf{id}^{(j)}$ and $pk_n^{(j)}$ are unlinkable.

6.4 Implementation and Deployment

We now present an implementation of the decentralized registration systems described in Sections 6.3.1 and 6.3.2; i.e., a system that allows for decentralized registration in both a standard (in which no privacy is achieved) and blinded (in which privacy is achieved) fashion. Our implementation is built on top of SCPKI [143], which implements a basic web of trust system on the Ethereum blockchain. We extended SCPKI to support a blinded web of trust.

6.4.1 Overview

We have developed an identity management system based on blind signatures and deployed it on the Ethereum blockchain as a smart contract. The system allows users to sign the attributes of other users in a web of trust, and also allows a more official registration authority to sign a user attribute when needed (e.g., a public key signed by an identity registrar). In addition to the standard signing of generic attributes, public key attributes can also be signed in an anonymous way using blind signatures.

As in SCPKI, each user has their own identity on the blockchain that corresponds to an Ethereum address. Using the methods of the smart contract, users can add attributes to their Ethereum address, sign attributes, and revoke signatures. The system also provides a way for users to search and retrieve attributes, by producing Ethereum events, which allow clients to efficiently watch the blockchain for new changes by a smart contract.

Due to the expensive fees of Ethereum data storage, data associated with attributes may be stored off the blockchain but authenticated on the blockchain. This can be done by adding an address (e.g., a URI) for the location of the data instead of the data itself along with its cryptographic hash if necessary for authenticity. The smart contract allows for the ability to store data using IPFS (`https://ipfs.io/`) where the cryptographic hash of the data is also its address.

The signing and verification of signature validity is performed client-side. As described in Section 6.3.2, when checking a signature, the client must also look

for the existence of a revocation transaction as well as check the optional signature expiry date. Because of the incompatibility discussed in Section 6.3.2 between revocation and privacy, our implementation does not allow for the revocation of blind signatures — only standard signatures can be revoked.

6.4.2 Technical specification

The smart contract is written in Solidity, a high-level language for writing Ethereum contracts, and the client is written in Python. Our open-source implementation, based on SCPKI, consists of 1502 lines of Python and Solidity code. The client is a command line console application and provides access to the smart contract's methods and functionality to search for user attributes, retrieve attributes, retrieve signatures, and verify signatures. The smart contract sets the rules on the blockchain for the management and storage of identity attributes and keys (e.g., only the owner of a signature can publish a revocation for it). The smart contract, which consists of 46 lines of Solidity code, can be found on Github.[1] For the blind signature, we use the RSA blind signature scheme (as described in Section 6.1.2) with 2048-bit RSA keys.

Simple signing

In the setup phase, the user generates their own Ethereum address. To obtain a simple signature the user first adds an attribute to their Ethereum address, by calling the method **addAttribute** and specifying the attribute type and data. This creates an **AttributeAdded** event on the blockchain containing the attribute properties, which can be detected by the client. Because Ethereum events are indexable, the client can easily search for attributes. In the registration phase, the registrar signs the attribute by calling the method **signAttribute** and specifying the ID of the attribute to sign and optionally an expiry date of its signature. This creates an **AttributeSigned** event containing the signature properties, including the Ethereum address of the signer. Because only the owner of the private key of an Ethereum address can create transactions originating from that address, this cryptographically proves that

[1] `https://github.com/musalbas/trustery/tree/master/contract`

a specific Ethereum address signed an attribute. In the verification phase, the verifier checks the published signature on the user's attributes, checking that there are no revocations and that the signature has not expired.

Blind signing

If a user wants to obtain a blind signature in order to anonymously register a public key, they first publish a blinded public key attribute using the method **addBlindedAttribute**, providing the data for the blinded key and specifying the ID of the registrar's public key attribute on the blockchain that the key is blinded for; i.e., specifying which registrar the user wants to blindly sign the key. This creates a **BlindedAttributeAdded** event that can be detected by the owner of the signing public key attribute. To blindly sign an attribute, the registrar calls the method **signBlindedAttribute** on the blinded public key attribute previously added by the user, providing the data of the signature, this is done client-side. This creates a **AttributeBlindSigned** event. On receiving the event, the user can then unblind the signature client-side. In the verification phase, the user shows the unblinded signature to the verifier (as described in Section 6.3.2).

6.4.3 Costs

In Ethereum, every operation has a cost paid using gas. As of May 2017, this cost could be translated into ether and USD using the exchange rate of 1 gas = 0.00000002 ether, and 1 ether = $192.00.

Table 6.2 shows the cost of each operation when data is stored on and off the blockchain. Aside from the observation that operations are relatively cheap — publishing the contract is the most expensive step, at about $3, and all of the operations involving individual attributes cost a few cents — we also see that the operations that involve adding and signing attributes are significantly cheaper when the data representing attributes and blind signatures is stored on IPFS.

6.5 Conclusion

We have proposed different methods for achieving registration in public distributed ledgers. We presented a decentralized setting, where registration is potentially flex-

operation	gas	ether	USD
publish contract	786,586	0.0157	3.01
add standard RSA attribute	70,952	0.0014	0.27
add standard RSA attribute (IPFS)	40,713	0.0008	0.15
sign standard attribute	49,904	0.001	0.19
revoke standard attribute	28,514	0.0006	0.12
add blinded RSA attribute	60,173	0.0012	0.23
add blinded RSA attribute (IPFS)	38,303	0.0008	0.15
sign blinded RSA attribute	58,012	0.0012	0.23
sign blinded RSA attribute (IPFS)	36,079	0.0007	0.13

Table 6.2: Cost for operations, where all data is stored on the blockchain.

ible and can be done by several entities. For each case we presented the trade-offs between security (in the form of privacy and integrity), usability (in the form of passive or active verification), and efficiency. Moreover, all our solutions use only lightweight cryptographic primitives, as opposed to approaches that adopt zero-knowledge proofs or other advanced cryptography. We have also implemented a decentralized registration process that operates on the Ethereum blockchain and evaluated its costs and efficiency.

Our system does not provide a mechanism for key recovery, but we view this as an important open problem and an avenue for future research, especially in the setting in which a user has accumulated many signatures on an attribute and built up a robust on-chain identity.

Chapter 7

Conclusion and Open problems

A decentralized system that has one of its components centralized (e.g., governance) is still at risk of suffering similar abuse as centralized systems. For example, if the three biggest Bitcoin mining pools were to collude, they could decide to censor some transactions (or decide other arbitrary rules), putting the other miners and users in a situation where they either abide by those rules or fork the currency, which would require many efforts and the need to re-create a new community. To be truly decentralized, a system should have all of its components decentralized. As such a cryptocurrency where the consensus protocol has a high barrier to entry, where the decision-making process is not fully open or transparent or where the applications built on top of it require central authorities cannot really be considered decentralized.

In this thesis, we looked at cryptocurrencies and their level of (de)centralization. First, we noted that centralization can occur on different levels: (1) the consensus level; (2) the governance and (3) the application layer (for additional use cases). In order to solve (1), we presented in Chapter 4 a new consensus protocol, Fantômette, based on proof-of-stake. Because the barrier to entry to proof-of-stake is cheaper than proof-of-work and does not require physical investment, it is by nature less prone to centralization. Furthermore, it is significantly more energy-efficient than Bitcoin. To study (2), we presented in Chapter 5 a quantitative analysis of the centralization of the governance structure of Bitcoin and Ethereum. This allows us to meaningfully compare the two cryptocurrencies and

understand the dynamic of the discussion behind their decision-making. Finally (3), motivated by additional applications built on top of decentralized ledgers that may require registration of pseudonyms, we presented in Chapter 6 decentralized registration protocols.

The topic of decentralization of cryptocurrencies is a far-flung one and there are many avenues for future work for this thesis. On the topic of consensus for example, the problem of how to incorporate external incentives to the design of cryptocurrencies is, as of this writing, an open problem. Indeed as many different cryptocurrencies currently exist (more than two thousand as of this writing [23]) this creates some additional vectors of attack [71]. On the topic of governance a similar quantitative study as ours but on a different medium (e.g., Reddit, mailing lists, IRC) could be interesting. Additionally, let's note that the governance of more traditional open-source systems has been extensively studied in the social sciences literature from a theoretical point-of-view [144, 145, 146, 147]. However the economic component of cryptocurrencies makes their governance structure quite different as economic incentives are explicitly at stake. It would be interesting to have some theoretical and empirical comparative studies. Finally, on the application layer of blockchains there is the topic of how blockchains can be used to help provide transparency to systems that inherently have some central component.

Bibliography

[1] Alyssa Hertig. Major Blockchains Are Pretty Much Still Centralized, Research Finds, 2018. https://www.coindesk.com/major-blockchains-pretty-much-still-centralized-research-finds.

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

[3] Andy Greenberg. Visa, Mastercard move to choke WikiLeaks, December 2010. http://www.forbes.com/sites/andygreenberg/2010/12/07/visa-mastercard-move-to-choke-wikileaks/.

[4] Peter Oborne. Why did HSBC shut down bank accounts? https://www.bbc.co.uk/news/magazine-33677946, 2015.

[5] Jasmin Klofta and Tom Wills. Financial surveillance. https://media.ccc.de/v/34c3-9070-financial_surveillance, 2017.

[6] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *Proceedings on Privacy Enhancing Technologies*, 2017(4):404 – 426, 2017.

[7] Leslie Lamport. Paxos made simple. pages 51–58, December 2001.

[8] Miguel Castro, Barbara Liskov, et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[9] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 251–260. Springer-Verlag, 2002.

[10] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[11] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

[12] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*, pages 515–532, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[13] Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of bitcoin without the block reward. In *CCS '16*.

[14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2014.

[15] Sarah Azouvi, Mustafa Al-Bassam, and Sarah Meiklejohn. Who am I? Secure identity registration on distributed ledgers. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 373–389. Springer, 2017.

[16] Mike Caldwell and Aaron Voisine. *Passphrase-protected private key*, 2016.

[17] Sarah Azouvi, Mary Maller, and Sarah Meiklejohn. Egalitarian Society or Benevolent Dictatorship: The State of Cryptocurrency Governance. In *22nd International Conference on Financial Cryptography and Data Security*, 2018.

[18] Sarah Azouvi, Alexander Hicks, and Steven J Murdoch. Incentives in Security Protocols. In *Cambridge International Workshop on Security Protocols*, pages 132–141. Springer, 2018.

[19] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. In *Advances in Financial Technologies*, 2019.

[20] Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Betting on blockchain consensus with fantomette. *arXiv preprint arXiv:1805.06786*, 2018.

[21] Sarah Azouvi, Patric McCorry, and Sarah Meiklejohn. Winning the caucus race: Continuous leader election via public randomness. *arXiv preprint arXiv:1801.07965*, 2018.

[22] Sarah Azouvi and Alexandre Hicks. SoK: Tools for Game Theoretic Models of Security for Cryptocurrencies. *arXiv preprint arXiv:1905.08595*, 2019.

[23] CoinMarketCap. Cryptocurrency market capitalizations. `https://coinmarketcap.com/`. Accessed: 2018-01-15.

[24] Blockstream. `https://blockstream.com/`.

[25] Pete Rizzo. Blockstream raises $55 million to build out Bitcoin's blockchain, February 2016. `http://www.coindesk.com/blockstream-55-million-series-a/`.

[26] A next-generation smart contract and decentralized application platform, 2014. `https://github.com/ethereum/wiki/wiki/White-Paper`.

[27] Parity. https://parity.io/.

[28] Bips github. github.com/bitcoin/bips.

[29] Cecille De Jesus. The dao heist undone: 97% of eth holders vote for the hard fork, July 2016. https://futurism.com/the-dao-heist-undone-97-of-eth-holders-vote-for-the-hard-fork/.

[30] Ethereum classic. https://ethereumclassic.github.io/.

[31] Bitcoin cash. https://www.bitcoincash.org/.

[32] Bitcoin gold. https://bitcoingold.org.

[33] Bitcoin sv. https://bitcoinsv.io/.

[34] https://bitcointalk.org/index.php?topic=27787.0, 2011.

[35] Sunny King and Scott Nadal. PPCoin: Peer-to-peer crypto-currency with proof-of-stake, 2012. https://peercoin.net/assets/paper/peercoin-paper.pdf.

[36] Vlad Zamfir. Introducing Casper "the friendly ghost", August 2015. blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/.

[37] 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.

[38] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.

[39] Alexander Chepurnoy. Interactive proof-of-stake. *CoRR*, abs/1601.00275, 2016.

[40] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is bitcoin a decentralized currency? In *IEEE Security and Privacy*, 2014.

[41] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. Decentralization in bitcoin and ethereum networks. *Financial Cryptography and Data Security: 22nd International Conference*, 2018.

[42] Tyler Moore and Nicolas Christin. Beware the middleman: Empirical analysis of Bitcoin-exchange risk. In *Proceedings of Financial Cryptography and Data Security*, pages 25–33, 2013.

[43] Rainer Böhme, Nicolas Christin, Benjamin Edelman, and Tyler Moore. Bitcoin: Economics, technology, and governance. *The Journal of Economic Perspectives*, 29(2):213–238, 2015.

[44] Carla L Reyes, Nizan Geslevich Packin, and Benjamin P Edwards. Distributed governance, 2016.

[45] Neil Gandal and Hanna Halaburda. Can we predict the winner in a market with network effects? competition in cryptocurrency market. *Games*, 7(3):1–21, 2016.

[46] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. In *Proceedings of Eurocrypt*, 2018.

[47] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. IACR Cryptology ePrint Archive, Report 2016/919, 2016. `https://eprint.iacr.org/2016/919`.

[48] Lars Brünjes, Aggelos Kiayias, Elias Koutsoupias, and Aikaterini-Panagiota Stouka. Reward sharing schemes for stake pools. *arXiv preprint arXiv:1807.11218*, 2018.

[49] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 315–324. ACM, 2017.

[50] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Proceedings of Crypto*, 2018.

[51] Vitalik Buterin. Incentives in Casper the friendly finality gadget, 2017.

[52] BlackCoin. `https://blackcoin.co/`.

[53] NeuCoin. `http://www.neucoin.org/`.

[54] Tendermint. `https://tendermint.com/`.

[55] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. SPECTRE: A fast and scalable cryptocurrency protocol. IACR Cryptology ePrint Archive, 2016.

[56] Yonatan Sompolinsky and Aviv Zohar. PHANTOM: A scalable blockdag protocol. Cryptology ePrint Archive, Report 2018/104, 2018.

[57] Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Crypto. `https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV`.

[58] Lucianna Kiffer, Rajmohan Rajaraman, and abhi shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 729–744, New York, NY, USA, 2018. ACM.

[59] Joshua A. Kroll, Ian C. Davey, and Edward W. Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Proceedings of WEIS 2013*, 2013.

[60] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, 2013.

[61] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 305–320. IEEE, 2016.

[62] Christian Badertscher, Juan Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it work? a rational protocol design treatment of bitcoin. Cryptology ePrint Archive, Report 2018/138, 2018.

[63] Juan Garay, Jonathan Katz, Ueli Maurer, Bjoern Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. Cryptology ePrint Archive, Report 2013/496, 2013. `http://eprint.iacr.org/2013/496`.

[64] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *CCS '16*, CCS '16, pages 3–16. ACM, 2016.

[65] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Computer and Communications Security*, CCS '15, pages 706–719, New York, NY, USA, 2015. ACM.

[66] Itay Tsabary and Ittay Eyal. The gap game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 713–728, New York, NY, USA, 2018. ACM.

[67] Joseph Bonneau. Why buy when you can rent? In *Financial Cryptography and Data Security*, pages 19–26. Springer, 2016.

[68] Kevin Liao and Jonathan Katz. Incentivizing blockchain forks via whale transactions. In *International Conference on Financial Cryptography and Data Security*, pages 264–279. Springer, 2017.

[69] Yaron Velner, Jason Teutsch, and Loi Luu. Smart contracts make bitcoin mining pools vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 298–316. Springer, 2017.

[70] Jason Teutsch, Sanjay Jain, and Prateek Saxena. When cryptocurrencies mine their own business. In *International Conference on Financial Cryptography and Data Security*, pages 499–514. Springer, 2016.

[71] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2018.

[72] Joseph Bonneau. Hostile blockchain takeovers (short paper). In *BitcoinâĂŹ18: Proceedings of the 5th Workshop on Bitcoin and Blockchain Research*, 2018.

[73] F. Ritz and A. Zugenmaier. The impact of uncle rewards on selfish mining in ethereum. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 50–57, April 2018.

[74] Jianyu Niu and Chen Feng. Selfish mining in ethereum. *arXiv preprint arXiv:1901.04620*, 2019.

[75] Vitalik Buterin. Uncle rate and transaction fee analysis.

[76] Eric Budish. The economic limits of bitcoin and the blockchain. Technical report, National Bureau of Economic Research, 2018.

[77] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, JoÃńl Alwen, Georg Fuchsbauer, and Peter GaÅ¿i. SpaceMint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. `https://eprint.iacr.org/2015/528`.

[78] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solidus: An incentive-compatible cryptocurrency based on permissionless Byzantine consensus. *CoRR*, abs/1612.02916, 2016.

[79] Primavera De Filippi and Benjamin Loveluck. The invisible politics of bitcoin: governance crisis of a decentralised infrastructure. *Internet Policy Review*, 5, 2016.

[80] Cathy Barrera and Stephanie Hurder. Blockchain upgrade as a coordination game. 2018.

[81] Marcella Atzori. Blockchain technology and decentralized governance: Is the state still necessary?, 2015.

[82] Wessel Reijers, Fiachra O'Brolcháin, and Paul Haynes. Governance in blockchain technologies and social contract theories. *Ledger*, 1(0):134–151, 2016.

[83] Vili Lehdonvirta. The blockchain paradox: Why distributed ledger technologies may do little to transform the economy, 2016. `www.oii.ox.ac.uk/the-blockchain-paradox-why-distributed-ledger-technologies-may-`

[84] Balaji Srinivasan and Leland Lee. Quantifying decentralization, 2017. `https://news.earn.com/quantifying-decentralization-e39db233c28e`.

[85] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A decentralized public key infrastructure with identity retention. IACR Cryptology ePrint Archive, Report 2014/803, 2014. `http://eprint.iacr.org/2014/803.pdf`.

[86] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack Resilient Public-Key Infrastructure. In *Proceedings of ACM CCS 2014*, pages 382–393, 2014.

[87] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[88] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In

Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[89] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[90] Christina Garman, Matthew Green, and Ian Miers. Decentralized anonymous credentials. In *Proceedings of the NDSS Symposium 2014*, 2014.

[91] Thomas Hardjono and Alex (Sandy) Pentland. Verifiable anonymous identities and access control in permissioned blockchains, 2016. `http://www.mit-trust.org/s/ChainAnchor-Identities-04172016.pdf`.

[92] Consensys. uport: The wallet is the new browser. `https://medium.com/@ConsenSys/uport-the-wallet-is-the-new-browser-b133a83fe73`. Accessed: 2016-08-04.

[93] Philipp Schmidt. *Certificates, Reputation, and the Blockchain*, 2015.

[94] Hyperledger indy. `https://www.hyperledger.org/projects/hyperledger-indy`.

[95] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *arXiv preprint arXiv:1802.07344*, 2018.

[96] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, January 1983.

[97] Michael Ben-Or and Nathan Linial. Collective coin flipping, robust voting schemes and minima of banzhaf values. In *26th FOCS*, pages 408–416, Portland, Oregon, October 21–23, 1985. IEEE Computer Society Press.

[98] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *Proceedings of the IEEE Symposium on Security & Privacy*, 2017.

[99] M. Rabin. Transaction protection by beacons. Technical Report 29-81, Aiken Computation Laboratory, Harvard University, 1981.

[100] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015. `https://eprint.iacr.org/2015/1015`.

[101] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 120–130, 1999.

[102] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. `https://eprint.iacr.org/2018/712`.

[103] Debraj Ray. An infinite extensive form, 2006. `https://www.econ.nyu.edu/user/debraj/Courses/GameTheory2006/Notes/infinite.pdf`.

[104] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 53–62, New York, NY, USA, 2006. ACM.

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

[106] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. BAR Fault Tolerance for cooperative services. *SIGOPS Oper. Syst. Rev.*, 39(5):45–58, October 2005.

[107] Ittai Abraham, Lorenzo Alvisi, and Joseph Y. Halpern. Distributed computing meets Game Theory: Combining insights from two fields. *SIGACT News*, 42(2):69–76, June 2011.

[108] R. Palmieri. Leaderless consensus: The state of the art. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1307–1310, May 2016.

[109] Michael Saks. A robust noncrytographic protocol for collective coin flipping. *SIAM J. Discret. Math.*, 2(2):240–244, May 1989.

[110] Alexander Russell and David Zuckerman. Perfect information leader election in log*n+o(1) rounds. *J. Comput. Syst. Sci.*, 63(4):612–626, December 2001.

[111] Uriel Feige. Noncryptographic selection protocols. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 142–, Washington, DC, USA, 1999. IEEE Computer Society.

[112] Rafail Ostrovsky, Sridhar Rajagopalan, and Umesh Vazirani. Simple and efficient leader election in the full information model. In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 234–242, New York, NY, USA, 1994. ACM.

[113] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 990–999, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[114] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In *Applied Cryptography and Network Security*, Cham, 2017.

[115] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. Proofs-of-delay and randomness beacons in Ethereum. In *Proceedings of the IEEE S&B Workshop*, 2017.

[116] Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of wealth in proof-of-stake cryptocurrencies. *arXiv preprint arXiv:1809.07468*, 2018.

[117] Erica Blum, Aggelos Kiayias, Cristopher Moore, Saad Quader, and Alexander Russell. The combinatorics of the longest-chain rule: Linear consistency for proof-of-stake blockchains. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1135–1154. SIAM, 2020.

[118] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*, pages 1–10, Sept 2013.

[119] M Lynne Markus. The governance of free/open source software projects: monolithic, multidimensional, or configurational? *Journal of Management & Governance*, 11(2):151–163, 2007.

[120] Christoph Lattemann and Stefan Stieglitz. Framework for governance in open source communities. In *Proceedings of the 38th annual Hawaii international conference on system sciences*, pages 192a–192a. IEEE, 2005.

[121] Miltiadis Allamanis and Charles Sutton. Mining source code repositories at massive scale using language modeling. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 207–216. IEEE, 2013.

[122] Chadd C Williams and Jeffrey K Hollingsworth. Automatic mining of source code repositories to improve bug finding techniques. *IEEE Transactions on Software Engineering*, 31(6):466–480, 2005.

[123] Ralph Peters and Andy Zaidman. Evaluating the lifespan of code smells using software repository mining. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 411–416. IEEE, 2012.

[124] Yan Hu, Jun Zhang, Xiaomei Bai, Shuo Yu, and Zhuo Yang. Influence analysis of github repositories. *SpringerPlus*, 5(1):1268, 2016.

[125] Eips github. `github.com/ethereum/EIPs`.

[126] Richard Simard, Pierre LâĂŹEcuyer, et al. Computing the two-sided kolmogorov-smirnov distribution. *Journal of Statistical Software*, 39(11):1–18, 2011.

[127] S. Mehta. *Statistics Topics*. Createspace Independent Pub, 2014.

[128] Jasjeet S. Sekhon. Multivariate and propensity score matching software with automated balance optimization: The Matching package for R. *Journal of Statistical Software*, 42(7):1–52, 2011.

[129] T.âĂL'A. Sø rensen. Sørensen tj. a method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. biologiske skrifter/kongelige danske videnskabernes selskab 5: 1-34. 5:1–34, 01 1948.

[130] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[131] Tezos. `tezos.com`.

[132] Steemit. `steemit.com`.

[133] Lis Evenstad. Dwp trials blockchain technology for benefit payments. `http://www.computerweekly.com/news/450300034/`

DWP-trials-blockchain-technology-for-benefit-payments. Accessed: 2016-08-04.

[134] Rory Cellan-Jones. Blockchain and benefits - a dangerous mix? http://www.bbc.com/news/technology-36785872. Accessed: 2016-08-04.

[135] Gill Plimmer. Use of bitcoin tech to pay UK benefits sparks privacy concerns. http://www.ft.com/cms/s/0/33d5b3fc-4767-11e6-b387-64ab0a67014c.html.

[136] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA.

[137] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography. http://cseweb.ucsd.edu/~mihir/papers/gb.pdf, 2000.

[138] United States Postal Service. Federal cloud credential exchange (fccx), August 2013. https://www.fbo.gov/spg/USPS/SSP/HQP/1B-13-A-0003/listing.html.

[139] UK Cabinet Office and Government Digital Service. Introducing GOV.UK Verify, September 2015. https://www.gov.uk/government/publications/introducing-govuk-verify.

[140] Luís T. A. N. Brandão, Nicolas Christin, George Danezis, and Anonymous. Towards mending two nation-scale brokered identification systems. In *Proceedings on Privacy Enhancing Technologies*, 2015.

[141] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. SoK: The evolution of sybil defense via social networks. In *2013 IEEE Symposium on Security and Privacy*, pages 382–396, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[142] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[143] Mustafa Al-Bassam. SCPKI: A smart contract-based PKI and identity system. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 35–40, New York, NY, USA, 2017. ACM.

[144] Sonali K Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management science*, 52(7):1000–1014, 2006.

[145] Paul B De Laat. Governance of open source software: state of the art. *Journal of Management & Governance*, 11(2):165–177, 2007.

[146] Siobhán O?Mahony. The governance of open source initiatives: what does it mean to be community managed? *Journal of Management & Governance*, 11(2):139–150, 2007.

[147] Robert Viseur. Forks impacts and motivations in free and open source projects. *International Journal of Advanced Computer Science and Applications*, 3(2):117–122, 2012.