Securing Bitcoin-like Backbone Protocols against a Malicious Majority of Computing Power

Tuyet Duong* Lei Fan† Thomas Veale‡ Hong-Sheng Zhou§

July 21, 2016

Abstract

Cryptocurrencies like Bitcoin have proven to be very successful in practice and have gained lots of attention from the industries and the academia. The security of Bitcoin-like systems is based on the assumption that the majority of the computing power is under the control of honest players. However, this assumption has been seriously challenged recently and Bitcoin-like systems will fail when this assumption is broken.

We propose the first Bitcoin-like protocol that is secure in the presence of a malicious majority of computing power. On top of Bitcoin's brilliant ideas of utilizing the power of the honest miners, via their computing power together with blocks, to secure the blockchain, we further leverage the power of the honest users, via their coins together with transactions, to achieve this goal. In particular, we propose a novel strategy for selecting the best blockchain from many competing chains by carefully comparing coins in these blockchains. In addition, we rigorously prove important security properties of our protocol in an extension of the blockchain analysis framework by Garay et al [Eurocrypt 2015].

*Virginia Commonwealth University, duongtt3@vcu.edu.

†Shanghai Jiao Tong University, fanlei@sjtu.edu.cn. Work done while at Virginia Commonwealth University.

‡Virginia Commonwealth University, vealetm@vcu.edu.

§Virginia Commonwealth University, hszhou@vcu.edu.

Contents

1 Introduction **1**
- 1.1 Our Techniques . . . 1
  - 1.1.1 Basic design principles and our design ideas . . . 2
  - 1.1.2 High-level idea of our scheme . . . 2
- 1.2 Related Work . . . 4

2 Model **5**
- 2.1 GKL Model . . . 5
- 2.2 Our Extended Model . . . 6

3 Construction **6**
- 3.1 Blocks and Blockchains . . . 7
  - 3.1.1 Blocks . . . 7
  - 3.1.2 Blockchains . . . 7
- 3.2 Transactions . . . 8
  - 3.2.1 Transaction Basics . . . 9
  - 3.2.2 Transactions in a Blockchain . . . 10
- 3.3 Coins in a Blockchain . . . 11
  - 3.3.1 Bound Transactions . . . 11
  - 3.3.2 Coins in a Transaction, in a Block, and in a Blockchain . . . 13
  - 3.3.3 Chain Comparison . . . 14
- 3.4 Our Protocol . . . 15
- 3.5 Discussions . . . 17
  - 3.5.1 How to bootstrap from a mature blockchain? . . . 17
  - 3.5.2 Enforcing honest users to stick to a single blockchain . . . 18
  - 3.5.3 Increasing the honest cash flow . . . 19
  - 3.5.4 Considerations in Stake for Consensus . . . 19

4 Security Analysis **20**
- 4.1 Security Properties of Backbone Protocol . . . 20
- 4.2 Preliminaries . . . 20
- 4.3 The Common Prefix Property . . . 21
- 4.4 The Chain Quality Property . . . 26

A Supportting Material for Our Construction **30**
- A.1 Transaction flow . . . 30
- A.2 Coinbase transaction . . . 31

1 Introduction

Cryptocurrencies like Bitcoin [27] have proven to be very successful in practice. In recent news, dozens of banks have joined a consortium to explore blockchain technology as a mode of transacting funds [1]. An effort by industry leaders and the Linux Foundation [37] to advance blockchain technology into a widely available software development tool is underway. On a national scale, the NSF funds millions in blockchain research [29].

At the heart of the Bitcoin system is a global public distributed ledger, called *blockchain*, that records all transactions between *users*. Bitcoin users create accounts to own bitcoins, and transactions enable a credit for one account and a corresponding debit to another account (i.e., transferring a certain number of bitcoins from a debit account to a credit account). The blockchain is established by a chain of cryptographic puzzles (also called proofs-of-work (PoWs) [6, 15]), solved by a peer-to-peer network of nodes called Bitcoin *miners*. Each miner that successfully solves a cryptographic puzzle is allowed to extend the blockchain with a block of transactions, and at the same time to collect a reward. The more computing power a miner invests, the better his chances are at solving the puzzle first.

To maintain the security of the Bitcoin blockchain, it requires that the majority of computing power is under the control of honest miners. Very recently, Garay et al [19] looked into the "core" of the Bitcoin system that they call the *backbone protocol* and for the first time introduce a formal framework with a rigorous security analysis. They showed several security properties of the Bitcoin blockchain assuming the majority of computing power is honest.

Unfortunately, the majority of honest mining power assumption could be seriously challenged. For example, the mining process could be dominated by a smaller number of professional miners with customized hardware making it unprofitable for others to join — this raises the natural concern that a few professional miners might collude to alter the contents stored in the blockchain [28]. Indeed, the mining pool GHash.io exceeded 50% of the computational power in Bitcoin in 2014 [20]. Efforts have been made to address this crisis. For example, memory hard puzzles are developed to curtail specialized hardware production [2–4, 8, 12]. Moreover, new strategies are designed to discourage the formation of mining pools [26]. Nevertheless, these techniques cannot help secure the system if the majority of computing power is malicious. We here ask the following question:

> *Is that possible to design a Bitcoin-like system which is secure against 51% malicious computing power?*

As discussed, this is a very important and challenging question to be answered. Cryptocurrency systems are very fragile in their early stage, and it will be extremely difficult, if not impossible, to "grow" a stable, Bitcoin-like blockchain if the adversary controls the majority of computing power at the very beginning. In this work, we do not offer a complete solution to the above question. Instead, we consider the following scenario where *a cryptocurrency system has "grown up" and then the adversary suddenly controls the majority of computing power to attack on this already mature system*. We argue that this scenario is realistic: it is very possible that the current Bitcoin system becomes unstable because some miners develop novel mining techniques and dominate the system for a short time period.

Proof-of-Stake (PoS) Blockchains [23, 24] might be promising approaches to defend against the malicious computational majority; however, most PoS blockchains suffer from typical "Nothing-at-Stake" attack, or is based on problematic initialization phase; furthermore, there is no rigorous security analysis to demonstrate that these systems are secure in this scenario. Nevertheless, jumping ahead, in this work, we answer the posed question by proposing a PoS-PoW-hybrid blockchain protocol called "modified-backbone protocol", and then provide a formal security analysis of it.

1.1 Our Techniques

For starters, we will introduce our design philosophy and some motivation for this work, then we will move into the fundamental ideas behind our scheme and why it works.

1.1.1 Basic design principles and our design ideas

Avoiding heavy tools and costly external resources. There are many reasons that Bitcoin has become a successful system. For example, in the original design of Bitcoin system, heavy cryptographic tools are avoided — only standard cryptographic primitives such as hash functions and digital signature schemes are utilized. When we make an effort to design more robust blockchains as in this work, one basic principle that we follow is to use only lightweight cryptographic tools. The security of Bitcoin has been rigorously analyzed (see [19]) under the assumption that the majority of computing power is controlled by honest players (along with standard network assumptions). One may attempt to introduce external trustworthy resources to strengthen the blockchain; but we have to be very cautious since external resources are often expensive, which will probably lead to impractical implementations. Therefore, in this work, our second basic principle is to leverage the trustworthy resources within Bitcoin-like systems, and to avoid any external resources.

Leveraging the power of users, coins, and transactions. With the above basic principles in mind, we first need to identify certain "additional" resources *within the system*, to help the honest miners to protect the blockchain against the adversary who controls the majority of computing power. Where can we obtain such resources? Recall that in the Bitcoin-like system, two types of players, the *users* and the *miners*, are involved. Intuitively, miners control the computing resources, and users control the second type of resources, coins, in the system. In Bitcoin, brilliant design ideas enable utilizing honest miners, via their computing power, together with the blocks they generated, to secure the blockchain system. In this project, on top of the design ideas in Bitcoin, we will further leverage the power of the users, via their coins, together with transactions they generate, to achieve this goal. We argue that the second type of resources, coins, can be utilized since rational users are very willing to join the effort to "protect" the system (and their own coins).

Bootstrapping from certain existing cryptocurrency system. As we discussed above, if we can use suitable alternative resources in a system, like coins, then it might be possible to achieve our goal of securing the system against short time period but large-scale computational attacks. However, if the coins are distributed in a bad fashion, and the adversary is able to control a significant portion of these alternative resources, then it may be still difficult for us to achieve our goal. Nevertheless, we notice that there are systems (e.g., Bitcoin) which already "absorbed" a huge amount of honest computing power and distributed coins among lots of good miners. That huge amount of honest resources could be used as a buffer to defend against an adversary who can collect lots of computing power in a short time period. Indeed, our goal in this paper is to "bootstrap" from mature cryptocurrency systems and make them more robust.

1.1.2 High-level idea of our scheme

Again, our goal is to secure an already mature cryptocurrency system (such as nowadays Bitcoin) against the malicious majority of computing power. In such a mature cryptocurrency system, the majority of coins is controlled by honest users[1]. Therefore, we require honest users to utilize their resources (i.e., coins) to help select the best blockchain. How can honest users utilize their coins to support their best chain? A possible way for an honest user to go is to include the user's best available blockchain as part of the transaction; this transaction is then signed by the user and *bound* to the user's current best available blockchain. When this transaction is broadcast into the network, all miners are informed with the user's best choice of blockchain. After collecting all users' best choices, miners will be able to derive and extend the best blockchain. Jumping ahead, in section 3.2.1, we slightly modify the Bitcoin transaction format so that the users can support their best chains; we note that such modification will not increase the effort of users.

The next question is how to use coins to *define* the best chain. This is a tricky question. Recall that the adversary may control the majority of computing power; thus, choosing the most difficult chain as the best blockchain (as in Bitcoin) is not a good strategy anymore. We need to select the best blockchain based on a different strategy. As we discussed in the previous paragraph, each transaction is bound to a blockchain that is selected by a user, which allows us to define the balance of the output/credit account in the transaction as

[1] For simplicity of our security analysis, we assume that no new coins are generated and all coins originate from a mature blockchain.

the "supporting coins" to endorse the blockchain. The immediate idea is to compare the supporting coins of two different blockchains, and then decide which one is better. However, this idea doesn't work directly: the adversary is able to create a blockchain with a huge amount of supporting coins by simply spending his coins repeatedly and frequently along the blockchain. We can avoid this issue by only counting the unspent coins which have not been transferred to any account in this blockchain. However, this strategy is still problematic, and we cannot count the unspent supporting coins from the beginning of the blockchain; note that the total number of supporting coins in each blockchain is the same since the coins are only moved from accounts to accounts. To address this issue, our final strategy is to *compare the unspent supporting coins in two branches from the block where the chains diverge (divergent position)*. The branch with more unspent supporting coins is defined as the better chain. Please see Appendix A.1 and Figure 4 for more explanation. In section 3.3, we carefully define the best chain selection strategy based on coins in blockchains.

Finally, in section 3.4, we present the details of our protocol. The protocol consists of two types of rounds executing one by one: *miner-round* and *user-round*. In each round, each player (miner or user) first computes its best valid chain based on the selection strategy described above; then the player attempts to extend the chain via proof of work (or issues a transaction); finally the player broadcasts its attempt or computed transactions to the network.

Why the scheme works?[2] The desired security properties we want to prove for the scheme are *common prefix* and *chain quality* properties [19]. Basically, the common prefix property means that any two honest parties' best blockchains are agreed except the most recent a few blocks. The chain quality property implies that, given any set of consecutive blocks on the best blockchain of an honest user, the ratio of the blocks that are contributed by malicious miners is suitably bounded. As in [19], our protocol is carried out in a synchronous[3] but not authenticated network. We assume the honest miners control only a small constant portion of mining power; in addition, honest users control the majority of coins, and they spend enough number of coins in each round to provide their support to the best blockchain.

To facilitate the presentation of our proof ideas, we introduce the notion of a *pure successful* round. We say a protocol execution round is pure successful if in that round at least one honest miner finds a puzzle solution and no malicious miner finds any. Since honest miners control a constant ratio of the total mining power, pure successful rounds can be reached frequently. In the pure successful round, the newly extended blockchain contains all of the supporting coins collected by an honest miner. Any blockchain that is extended before the pure successful round cannot contain the coins spent in the current pure successful round. That is, the newly extended blockchain will have more coins than any "older" blockchain. Then, the best blockchain in the view of every honest user will align, which is the extended blockchain after a pure successful round. We also note that when the system reaches an aligned round, the malicious miners cannot fork a valid branch from any position before the newly extended block. In summary, the adversary can only create divergent views for different miners between two pure successful rounds. The malicious miners cannot find too many new blocks between two pure successful rounds. Indeed, this leads to our proof for the common prefix property. The detailed proof can be found in Section 4.3.

Now we argue how to achieve the chain quality property. As we mentioned in the previous paragraph, in each pure successful round, an honest miner will contribute a block to the best blockchain. For any consecutive rounds, the number of pure successful rounds will follow a Binomial distribution. At the same time, the number of blocks generated by malicious miners will also follow a Binomial distribution. We therefore can have a lower bound on the number of blocks that are contributed by honest miners, and an upper bound on the number of blocks that are contributed by malicious miners. This will lead us to show the chain quality property: the ratio of the blocks that contributed by malicious miners is suitably bounded. The detailed proof can be found in Section 4.4.

Besides the analysis in our cryptographic analysis framework (see Section 4), it would be desirable to define

[2]This is the first rigorous cryptographic analysis of Bitcoin-like protocols that are resilient to a malicious majority of computing power.

[3]It is nontrivial to get a solution in the synchronous setting. Previously, [5,21] provided solutions to (non-practical) consensus; the goal here is to get a *practical* blockchain against malicious 51%, which is much harder.

a comprehensive analysis framework in the rational setting. Unfortunately, there is no known analysis framework capturing comprehensive rational attacks, and this is one of the major open questions in the field. Nevertheless, in Section 3.5.2, we provide informal arguments that our constructions can defend against typical attacks such as "Nothing-at-Stake". Moreover, based on the best chain strategy we defined (see Section 3.3.3), it is easy to see our protocol is immune to the Sybil Attack [14]. We further note that, given the fact that our best chain is based on the number of supporting coins, the adversary cannot increase the value of his favorite chain by passing coins back and forward along that chain.

1.2 Related Work

Anonymous digital currency was introduced by Chaum [10] in early 1980s. The first decentralized currency system, Bitcoin [27], was launched about 30 years later, by incentivizing a set of players to solve cryptographic puzzles (also called proofs-of-work puzzles [6, 15]).

It is very difficult to provide rigorous analysis for Bitcoin like protocols. Eyal and Sirer analyze Bitcoin protocol in the rational setting [18], revealing that Bitcoin falls short of incentivizing correct behavior among miners. The mining ecosystem has since been extensively studied in [17, 30]. Garay et al. [19] propose the first comprehensive *cryptographic* framework and show that, assuming the honest majority of computing power, Bitcoin achieves certain interesting security notions they defined. Their positive results fall apart if the adversaries' computational resources are not suitably bounded (strictly less than half). In fact Eyal and Sirer show in [18] that rational adversaries require much less for a disproportionate advantage. In this work, we propose a new protocol which can circumvent the strict honest majority assumption of computing power by using native system resources (i.e., coins) for making consensus decisions; along the way, we introduce an extended framework of [19] which allows us to analyze the security of our new protocol. While Garay et al.'s and our own are in the synchronous setting, we remark that very recently, Pass et al [32] provide the first cryptographic analysis of the blockchain protocol in the asynchronous setting.

The most related to our design strategy is a class of consensus algorithms generally known as *proof-of-stake* (PoS). We note that there is no formal cryptographic analysis that shows PoS can overcome the strict majority assumption of computing power (even some claimed so). A proof-of-stake is a de facto proof-of-ownership of an account that controls some assets. These assets are then used for something meaningful, such as adding or validating blocks. The two PoS variants that are closest to our consensus method are Peercoin [23] and an (unimplemented) idea contributed on the Bitcoin Wiki [36] by Meni Rosenfeld. Peercoin is the first public system to implement PoS [23] that leverages a transaction prioritizing feature from Bitcoin called *coinage*. This priority value is the product of the number of coins in a transaction and the time elapsed since they were last transferred. From this, the Peercoin best chain policy is the chain (and all of its transactions) with the most coinage consumed. Rosenfeld [36] proposes that some blocks in a chain are special signing blocks, where the owners of coins may sign and broadcast those blocks with proofs that they own some coins. These signatures count as votes for the blockchains they want to follow. In this way, consensus is based on the chains with the most votes from the users.

Unfortunately, neither of these strategies work on their own. Peercoin suffers from many issues related to coinage. Consider a user that owns a small fraction of coins in a few accounts and stores up a lot of coinage. Then, they can use those accounts to produce multiple blocks in a row and execute a double-spend attack. Several ad-hoc attempts have been made to improve on the basic PoS technique such as tuning the rate of coinage accumulation [13, 34], but there has yet to be a formal proof demonstrating that these are effective strategies. On top of this, Peercoin relies on a centralized check-pointing system[4]. Rosenfeld's best chain policy is not clear enough and this leaves a loophole for an adversary with a majority of computing power and a small fraction of coins to segment the network and balance the chains, which would violate consensus. Our contribution addresses the issues that arise from both of these approaches and we supply a formal proof of security. Besides Peercoin and Rosenfeld's proposal, there are several proposals solved some of the issues such as Tendermint [24] and

[4]While this checkpoint system is no longer a critical part of the system, there are still periodic checkpoints introduced by the development team like in Bitcoin.

Ethereum's Casper [38], which can circumvent the nothing at stake problem; however, those constructions indeed face other obstacles. Please see Section 3.5.4 for detailed discussions.

On another front, attempts to limit a single entity from controlling too much computing resource is being made through complex, ASIC (Application Specific Integrated Circuits) resistant, and memory hard puzzles [2–4,8,12] as an alternative to Bitcoin's SHA-256 proof-of-work puzzle. Other attempts aim to prevent pools from forming via non-outsource puzzles [26]. In addition, other important approaches [16,25,31] have been explored by the research community.

Organization. In Section 2, we present the framework of Garay et al [19], and then extend it for our purpose. In Section 3, we present the details of our construction. Then, in Section 4 we analyze the security of our construction. Finally, additional supporting materials for our construction are provided in Appendix A.

2 Model

2.1 GKL Model

In order to study the security of Bitcoin-like protocols, Garay et al [19] consider a standard multi-player synchronous communication setting (e.g., Canetti's formulation of the "real world" execution [9]) with the relaxation that *(i)* the underlying communication graph is not fully connected, and *(ii)* the underlying communication channel is reliable but not authenticated. Note that the adversary is not allowed to stop the messages from being delivered, but the adversary may "spoof" the source of a message they transmit and impersonate the (honest) sender of the message. BROADCAST is used to denote the atomic unauthenticated "send-to-all" functionality in this synchronous communication model; here, an adversarial sender may abuse BROADCAST by sending inconsistent messages to different honest miners to confuse them. In this model, the adversary is "rushing" in the sense that in any given round the adversary is allowed to see all honest miners' messages before deciding his strategy. We finally note that due to the unauthenticated nature of the communication model, the miners may never be aware of the exact number of participants in a protocol execution. For simplicity and similar to the framework by [19], we assume that the number of parties is fixed during the course of the protocol execution.

Miners. The environment $\mathcal{Z}$ provides inputs for all miners and receives outputs from them. The environment may provide input to any miner in any round, and may modify the input from round to round. We denote by INPUT() the input tape, by OUTPUT() the output tape, and by RECEIVE() the incoming communication tape, of each miner.

In each round, miners read their input tape INPUT() and communication tape RECEIVE(), then carry out some computation, and then send out a BROADCAST message which is guaranteed to be delivered to all miners in the beginning of the next round. Let Π be an n-party protocol among miners $P_1,\ldots,P_n$. Let $\{\text{VIEW}^{P_i}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa,z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ denote the random variable ensemble describing the view of miner P_i after the completion of an execution with environment $\mathcal{Z}$, running protocol Π, and adversary $\mathcal{A}$, on auxiliary input $z\in\{0,1\}^*$, and the security parameter κ. For simplicity, the parameters κ and z are often dropped if the context is clear, and we describe the ensemble by $\text{VIEW}^{P_i}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa,z)$. The concatenation of the views of all miners $\langle\text{VIEW}^{P_i}_{\Pi,\mathcal{A},\mathcal{Z}}\rangle_{i=1,\ldots,n}$ is denoted by $\text{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}$.

We note that the protocol's miners may never be certain about the number of participants in the protocol execution, given the unauthenticated nature of the communication model. Moreover, for simplicity, only a standalone static model is considered in this model, and the number of miners is fixed during the course of the protocol execution.

Bounded computing power. In Bitcoin, the miners have limited ability to produce proofs of work. To capture this, in [19], all miners are assumed to have access to a random oracle $H(\cdot)$, and each miner is allowed to perform at most q queries in each round; here, q is a function of the security parameter κ, and we also call such miners *q-bounded*. Note that this is an "idealized" interpretation of the setting where all miners have the same amount of computing power. However, in the reality, each different honest miner may have a different amount of computing power; nevertheless, this idealized-model does not sacrifice generality since one can imagine that real honest

miners are simply clusters of some arbitrary number of honest idealized-model miners. The adversary $\mathcal{A}$ is allowed to perform $t \cdot q$ queries per round, where t is the number of corrupted miners.

2.2 Our Extended Model

In this extended model, we use the same model for the communications and the miners. We also use a synchronous unauthenticated communication system with q-bounded miners. However, we introduce the users into the model as follows.

Users and Miners. In our extended model, we specify two types of rounds that execute in turn: user-round and miner-round. In the user-round, the users are able to create accounts and sign transactions using their private keys. Moreover, they can utilize their coins to help the honest miners secure the system. On the other hand, in the miner-round, the miners solve PoW puzzles, and append a new block to the best chain on their local views. The users and miners are playing different roles in our model; however, without the collaboration of these two types of players, our model cannot be secure in the presence of the malicious majority computing power.

Honest cash flow. The user is another type of players in the system who does not solve the POW to extend the blockchain, but spending coins in each round by signing transactions. A transaction is a statement to show that some coins are transferred from a source account to some destination accounts. A continuous transactions can be viewed as a chain of transactions. We only concern with the coins of destination accounts since the coins of the source account will become invalid once they are transferred. We assume that there is an initial blockchain, denoted $\mathcal{C}^{\emptyset}$, which is distributed to all users before running the protocol. All of the legal coins must be traced back to $\mathcal{C}^{\emptyset}$ by following the reverse order of the corresponding chain. Furthermore, a transaction must be verifiable and a coin cannot be transferred twice from the same source.

The users adopt the same communication assumption as the miners. For simplicity, we consider the protocol executes round-by-round. We assume that the malicious users control at most $c_{\text{malicious}}$ coins, and in each round the honest users transfer $c_{\text{spend}} = \Theta(\kappa) \cdot c_{\text{malicious}}$ coins that originate from the blocks which were generated in $\Theta(\kappa)$ rounds earlier where κ is the security parameter. Note that c_{spend} denotes the *cash flow volatility* (v for short) of honest users. An honest user will sign his transaction to show his support to the bestchain on his local view.

We define a new comparison method for comparing two blockchains. In particular, the best chain is chosen based on the number of supporting coins. Given two distinct blockchains, the chain with more supporting coins from users after they are forked is better than the other one. We remark that once a coin is transferred, the supporting power is also transferred.

We refer to the above restrictions on the environment, the players, and the adversary as the (q, v)-synchronous setting. The view of a player participating in the protocol is denoted by $\text{VIEW}^{P,H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$, and the concatenation of all players' views is denoted by $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$. Similar to the framework by Garay et al [19], we will study *security properties* (see Section 4) of protocols Π in the (q, v)-synchronous setting. Such properties will be defined as predictions over the random variable $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$ by quantifying over all possible adversaries $\mathcal{A}$ and environments $\mathcal{Z}$.

3 Construction

We start by defining blocks, blockchains, proofs of work, a chain validation algorithm, and other related notations in Section 3.1; we remark that many of the notations in this section are borrowed from [19]. Then in Section 3.2, we define transactions in our protocol; here, new transaction format and properties are introduced. In Section 3.3, we introduce new tools so that coins in a blockchain can be used to protect the blockchain. Finally, we present our main protocol in Section 3.4.

3.1 Blocks and Blockchains

3.1.1 Blocks

Let $G(\cdot)$ and $H(\cdot)$ be cryptographic hash functions with output in $\{0,1\}^\kappa$. A *block* B is a quadruple of the form $B = \langle s, X, ctr, w\rangle$ where $s \in \{0,1\}^\kappa$, $X \in \{0,1\}^*$, $ctr \in \mathbb{N}$, $w \in \{0,1\}^\kappa$, such that they satisfy the predicate $\mathsf{validblock}_q^D(B)$, defined as $(H(ctr, w, G(s, X)) < D) \wedge (ctr \leq q)$ where the parameters $D \in \mathbb{N}$ and $q \in \mathbb{N}$. Here, D denotes the *difficulty level* of the block, and q denotes the maximum allowed number of hash queries in a round. We allow the size of q to be arbitrary as per backbone specification, as it bounds any player's ability to solve proofs of work in a round. Note that here, s should be the digest of the previous block, and w is a random nonce. Finally, we call X the *payload* of a block, and denote it as $\mathsf{payload}(B)$ which refers to the information stored in some block B. Specifically, in our protocol, we are interested in keeping *transactions* in X.

3.1.2 Blockchains

A *blockchain* $\mathcal{C}$ consists of a sequence of ℓ concatenated blocks $B_1\|B_2\|\cdots\|B_\ell$, where $\ell \geq 0$. For each blockchain, we specify several notations such as head, length, and payload as below: *blockchain head*, denoted $\mathsf{head}(\mathcal{C})$, refers to the topmost block B_ℓ in blockchain $\mathcal{C}$; *blockchain length*, denoted $\mathsf{len}(\mathcal{C})$, is the number of blocks in blockchain $\mathcal{C}$, and here $\mathsf{len}(\mathcal{C}) = \ell$; finally, *blockchain payload*, denoted $\mathsf{payload}(\mathcal{C})$, refers to the information we store in $\mathcal{C}$, and we have $\mathsf{payload}(\mathcal{C}) = \|_{i=1}^{\ell} \mathsf{payload}(B_i)$. For brevity, we use sub blockchain (or subchain) for referring to segment of a chain. For example, $\mathcal{C}[1, \ell]$ refers to an entire blockchain, whereas $\mathcal{C}[j, m]$, with $j \geq 1$ and $m \leq \ell$ would refer to a sub blockchain $B_j\|\cdots\|B_m$. We use $\mathcal{C}[i]$ to denote the i-th block B_i in blockchain $\mathcal{C}$. If blockchain $\mathcal{C}$ is a prefix of another blockchain $\mathcal{C}'$, we write $\mathcal{C} \preceq \mathcal{C}'$.

Next, we discuss how to extend a blockchain, and how to validate a blockchain. In subsection 3.3.3, we will discuss how to compare blockchains.

Chain extension: Proof of work. The pow algorithm attempts to extend a chain via solving a proof of work (POW). This algorithm is parameterized by two hash functions $H(\cdot), G(\cdot)$ (which in our analysis will be modeled as random oracles); as well as two positive integers q and D where q represents the number of times the algorithm is going to attempt to brute-force the hash function inequality (see subsection 3.1.1) that determines the POW instance, and D determines the "difficulty" of the POW. The algorithm works as follows. Given a chain $\mathcal{C}$, an initial chain $\mathcal{C}^\emptyset$, and a value X to be inserted in the chain $\mathcal{C}$; the POW algorithm samples a random initial string w of length κ, it then hashes these values to obtain h with a counter ctr. Subsequently, it increments ctr and checks if $H(ctr, w, h) \leq D$; if a suitable (ctr, w, h) is found then the algorithm succeeds in solving the POW and extends chain $\mathcal{C}$ by one block. Note that the payload X, the initial string w, and the counter ctr will be inserted to the new block. If no suitable (ctr, w, h) is found, the algorithm simply returns the chain unaltered. We suppose that we have an initial blockchain $\mathcal{C}^\emptyset$ as the starting point, and $\mathcal{C}^\emptyset$ is known to all miners and users. We remark that, to ensure that each attempt is independent, we explicitly ask each miner to include a random string w in the proof of work.

Chain validation. The next algorithm, called validate_chain, is introduced to verify the structure of an input chain $\mathcal{C}$. This algorithm is nearly identical to the validate algorithm in [19] except that it only considers the chain that has a prefix $\mathcal{C}^\emptyset$ and a random nonce w is introduced. Algorithm validate_chain is parameterized by values q and D, as well as access to the hash functions $H(\cdot)$ and $G(\cdot)$. It first checks whether the considered chain $\mathcal{C}$ has prefix $\mathcal{C}^\emptyset$. If $\mathcal{C}^\emptyset$ is the prefix of $\mathcal{C}$, then starting from the head of $\mathcal{C}$, for every block, the algorithm validate_chain ensures that the corresponding proof of work is properly solved where the counter ctr does not exceed q, a hash pointer s of the previous block is properly referenced, and the proof of work satisfies the difficult D. If all blocks verify is true then the chain is valid, otherwise it is rejected. Please refer to Algorithm 2 for more details.

Algorithm 1: The *proof of work* function, parameterized by q, D and the hash function $G(\cdot)$, $H(\cdot)$. The input is $(X, \mathcal{C}, \mathcal{C}^{\emptyset})$.

function pow $(X, \mathcal{C}, \mathcal{C}^{\emptyset})$
  if $\mathcal{C}^{\emptyset} \preceq \mathcal{C}$ **then**
    $\langle s', X', ctr', w' \rangle \leftarrow \mathsf{head}(\mathcal{C})$
    $s \leftarrow H(ctr', w', G(s', X'))$
    $ctr \leftarrow 1$
    $w \leftarrow \{0,1\}^{\kappa}$
    $B \leftarrow \epsilon$
    $h \leftarrow G(s, X)$
    while $ctr \leq q$ **do**
      if $H(ctr, w, h) < D$ **then**
        $B \leftarrow \langle s, X, ctr, w \rangle$
        break
      $ctr \leftarrow ctr + 1$
    $\mathcal{C} \leftarrow \mathcal{C} \| B$
    return $\mathcal{C}$
  else
    return $\mathcal{C}^{\emptyset}$

Algorithm 2: The *chain validation predicate*, parameterized by q, D, and the hash functions $G(\cdot)$, $H(\cdot)$. The input is $(\mathcal{C}, \mathcal{C}^{\emptyset})$.

function validate_chain $(\mathcal{C}, \mathcal{C}^{\emptyset})$
  $b \leftarrow (\mathcal{C}^{\emptyset} \prec \mathcal{C})$
  if $b =$ `True` **then**
    $\langle s, X, ctr, w \rangle \leftarrow \mathsf{head}\ (\mathcal{C})$
    $s' \leftarrow H(ctr, w, G(s, X))$
    repeat
      $\langle s, X, ctr, w \rangle \leftarrow \mathsf{head}\ (\mathcal{C})$
      if $\mathsf{validblock}_q^D(\langle s, X, ctr, w \rangle) \bigwedge H(ctr, w, G(s, X)) = s'$ **then**
        $s' \leftarrow s$; `/* Retain hash value */`
        $\mathcal{C} \leftarrow \mathcal{C}[1, \mathsf{len}(\mathcal{C}) - 1]$; `/* Remove the head from` $\mathcal{C}$ `*/`
      else
        $b \leftarrow$ `False`
    until $(\mathcal{C} = \mathcal{C}^{\emptyset}) \bigvee (b =$ `False`$)$;
  return b

3.2 Transactions

In the Bitcoin-like cryptocurrencies, the users transfer their coins via issuing transactions. In this subsection we focus on transactions: we will describe the new transaction format that we need for our construction, we then explain how the transactions are generated, and then we will consider how a transaction can be validated in a blockchain.

3.2.1 Transaction Basics

Transaction generation and verification. Transactions are datagrams that are produced and consumed by the users. In a *transaction authentication scheme*, the users are able to create accounts and sign *transactions*. Formally, we define the syntax of transaction authentication scheme as follows:

Definition 3.1. *A* transactions authentication scheme TxAuth *consists of a tuple of algorithms* $(\mathsf{keygen}, \mathsf{sign}, \mathsf{verify})$*, with access to the random oracle* $G(\cdot)$*, where*

- $(\mathsf{a}^{\mathsf{in}}, \mathsf{sk}) \leftarrow \mathsf{keygen}(1^\kappa)$*:* keygen *is a key generation setup algorithm that takes as input* 1^κ *(*κ *is the security parameter) and outputs a pair* $(\mathsf{a}^{\mathsf{in}}, \mathsf{sk})$*. Here, account* a^{in} *consists of a verification key* vk *and the account address* $G(\mathsf{vk})$*; that is,* $\mathsf{a}^{\mathsf{in}} = (\mathsf{vk}, G(\mathsf{vk}))$*. In addition,* sk *is the corresponding secret key of* a^{in}*.*

- $\mathsf{tx} \leftarrow \mathsf{sign}(\mathsf{sk}, \tilde{\mathsf{tx}})$*:* sign *is a transaction signing algorithm that takes as input a secret key* sk *and a message* $\tilde{\mathsf{tx}}$ *in the following format:* $\tilde{\mathsf{tx}} = \left((\mathsf{a}^{\mathsf{in}}, \mathsf{c}^{\mathsf{in}}) \rightarrow (\mathsf{a}_1^{\mathsf{out}}, \mathsf{c}_1^{\mathsf{out}}), \ldots, (\mathsf{a}_\ell^{\mathsf{out}}, \mathsf{c}_\ell^{\mathsf{out}}), B\right)$ *where* $\mathsf{a}^{\mathsf{in}} = (\mathsf{vk}, G(\mathsf{vk}))$ *is the input account to be debited* c^{in} *coins;* $\mathsf{a}_i^{\mathsf{out}} = (\mathsf{vk}_i, G(\mathsf{vk}_i))$ *is the output account to be credited* $\mathsf{c}_i^{\mathsf{out}}$ *coins, for* $i = 1, \ldots, \ell$*; and* B *is a block. For simplicity, we use* $\tilde{\mathsf{tx}}.\mathsf{a}^{\mathsf{in}}$ *to access the input account of* $\tilde{\mathsf{tx}}$ *and we use* $\tilde{\mathsf{tx}}.B$ *to access the block* B *of* $\tilde{\mathsf{tx}}$*. Then* sign *outputs a message* $\mathsf{tx} = (\tilde{\mathsf{tx}}, \mathsf{vk}, \sigma)$ *where* σ *is the signature of* $\tilde{\mathsf{tx}}$*. For convenience, we call* $\tilde{\mathsf{tx}}$ unsigned *transaction and* tx signed *transaction.*

- $b := \mathsf{verify}(\mathsf{tx})$*:* verify *is a transaction verification algorithm that takes as input a message* $\mathsf{tx} = (\tilde{\mathsf{tx}}, \mathsf{vk}, \sigma)$ *and returns a bit* $b \in \{\mathtt{True}, \mathtt{False}\}$*. We say a transaction is* verifiable *if and only if* verify *returns* $\mathtt{True}$*.*

The security of the transaction authentication scheme is defined as follows:

Definition 3.2. *A transaction authentication scheme* $\Sigma = \langle \mathsf{keygen}, \mathsf{sign}, \mathsf{verify} \rangle$ *is* unforgeable *if for all* PPT *attackers* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{r}(\mathsf{a}^{\mathsf{in}}, \mathsf{sk}) \leftarrow \mathsf{keygen}(1^\kappa); (\tilde{\mathsf{tx}}, \sigma) \leftarrow \mathcal{A}^{\mathsf{sign}(\mathsf{a}^{\mathsf{in}}, \mathsf{sk}, \cdot)}(\mathsf{vk}); b \leftarrow \mathsf{verify}(\tilde{\mathsf{tx}}, \mathsf{vk}, \sigma) \\ : b = \mathtt{True} \text{ and } \mathcal{A} \text{ never submitted } \tilde{\mathsf{tx}} \text{ to } \mathsf{sign}(\mathsf{a}^{\mathsf{in}}, \mathsf{sk}, \cdot) \text{ oracle}\end{array}\right] \leq \mathsf{negl}(\kappa)$$

Remark 3.3. *Our transaction authentication scheme is very similar to the digital signature scheme in Bitcoin except that, the transaction* $\tilde{\mathsf{tx}}$ *to be signed here includes an additional field* B*. This modification is critical in our protocol construction in Section 3.4: there, we will use the additional field* B *to bind an unsigned transaction* $\tilde{\mathsf{tx}}$ *to a known block in a blockchain.*

account
+ verification_key vk;
+ account_address $G(\mathsf{vk})$;

credit
+ account $\mathsf{a}^{\mathsf{out}}$;
+ coin_number $\mathsf{c}^{\mathsf{out}}$;

unsigned_tx
+ account a^{in};
+ coin_number c^{in};
+ credit Cr;
+ block B;

tx
+ unsigned_tx $\tilde{\mathsf{tx}}$;
+ verification_key vk;
+ signature σ;

Figure 1: Data structure

Transaction format and data structure. We are now giving more details of our proposed data structures for account, credit, signed, and unsigned transactions in Figure 1. The account contains a verification key vk, and

its address $G(\mathsf{vk})$. The credit stores an output account and the number of coins c^{out}; thus, c^{out} coins will be transferred from an input account to a^{out}. The unsigned transaction includes a debited/input account and the number of debited coins c^{in} from the input account. Especially, this unsigned transaction also stores a field B to show its support to the corresponding block. The signed transaction consists of the corresponding unsigned transaction, a verification key, and the signature of this transaction.

In Figure 2, we present the format of a *regular transaction*. We employ the "dot" notation to indicate the access to an element of the data structure. For instance, if we want to access the block B in the data structure shown in Figure 2, we would refer to it as $\mathsf{tx}.\tilde{\mathsf{tx}}.B$. The signed transaction tx consists of three fields: a public key vk, an unsigned transaction $\tilde{\mathsf{tx}}$ for verification, and the signature σ of $\tilde{\mathsf{tx}}$. The unsigned transaction $\tilde{\mathsf{tx}}$ contains an input/debited account, the number of debited coins c^{in}, ℓ credited/output accounts, as well as the supported block B.

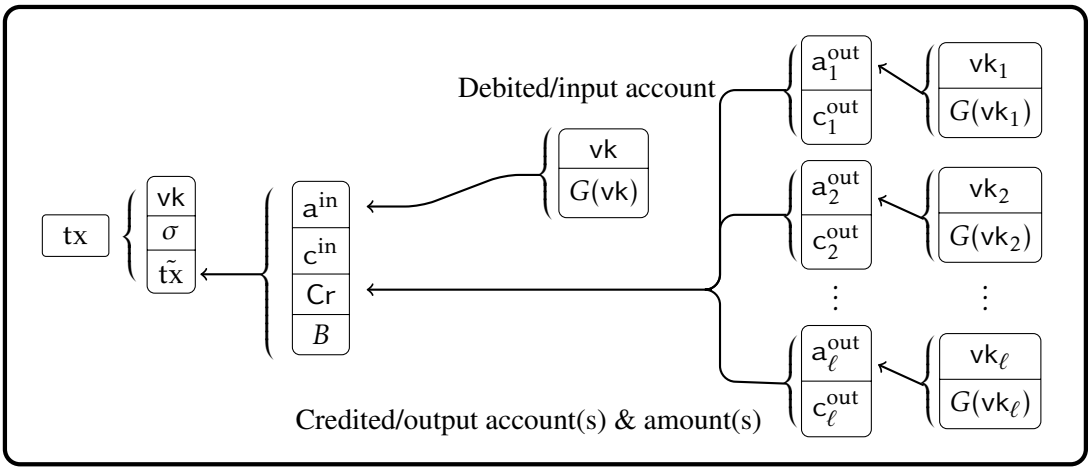

Figure 2: Transaction Data-structure Hierarchy, where $\mathsf{c}^{\text{in}} = \sum_{i=1}^{\ell} \mathsf{c}_i^{\text{out}}$.

We remark that *coinbase transactions* are not signed and these unsigned transactions may be included in a blockchain via proof of work (as in Bitcoin). A pictorial illustration of the format of a coinbase transaction can be found in the Appendix A.2.

3.2.2 Transactions in a Blockchain

After defining the basics of transactions, we now consider transactions in a blockchain. In this subsection, we define the transaction *well-formedness* in a blockchain. Intuitively, a well-formed transaction in a blockchain should be *verifiable* (as defined in Definition 3.1 above, i.e., verify predicate returns `True`), *traceable*, and *not conflicting*. Before giving the detailed definitions for traceable transactions and non-conflicting transactions, we note that without loss of generality, the initial state of a blockchain $\mathcal{C}$, denoted $\mathcal{C}^{\emptyset}$, consists of multiple coinbase transactions.

Traceable transactions. Transactions represent the ownership of coins in cryptocurrency systems such as Bitcoin. In our system, transactions have a one-to-many relationship that represents the transfer of coin(s) from a single input account to potentially many output accounts. In order to make sure that a new transaction is valid, we need to ensure that the input account of that transaction has a valid balance (i.e., c^{in}). More precisely, the balance is from a previous transaction; therefore, we must trace back one step to the previous transaction input. We do this until a transaction can be traced back to a legal source, specifically, a coinbase transaction in $\mathcal{C}^{\emptyset}$. In order to simplify our presentation, we assume that all accounts are unique and are used only once. We define a traceable transaction as follows.

Definition 3.4 (Traceable Transaction)**.** *Consider a blockchain $\mathcal{C} = \mathcal{C}^{\emptyset}\|B_1\|B_2\|\ldots\|B_l$ where $\mathcal{C}^{\emptyset}$ is the initial*

state and $l \in \mathbb{N}$, $\mathsf{payload}(\mathcal{C}^{\emptyset})$ *consists of a set of coinbase transactions, and* $\mathsf{payload}(B_1||B_2||\ldots||B_l)$ *consists of a set of regular transactions* $\{\mathrm{tx}_1,\ldots,\mathrm{tx}_m\}$ *where* $m \in \mathbb{N}$. *We say* tx *is a* traceable transaction *on* $\mathcal{C}$, *if there exist*
(1) a coinbase transaction $\mathrm{tx}'_0 \in \mathsf{payload}(\mathcal{C}^{\emptyset})$, *and*
(2) a subset of ordered transactions of $\{\mathrm{tx}_1,\ldots,\mathrm{tx}_m\}$, *denoted as* $\{\mathrm{tx}'_1,\ldots,\mathrm{tx}'_n\}$, *where* $n \in \mathbb{N}$ *and* $n \leq m$,
such that all of the following conditions hold:

- $\mathrm{tx} = \mathrm{tx}'_n$,
- $\forall i \in [n]$, $\exists t \in [\ell_i]$, *it holds that*
  (1), $\mathrm{tx}'_{i-1}.\tilde{\mathrm{tx}}.\mathsf{Cr}.\mathsf{a}^{\mathrm{out}}_t = \mathrm{tx}'_i.\tilde{\mathrm{tx}}.\mathsf{a}^{\mathrm{in}}$, *(2),* $\mathrm{tx}'_{i-1}.\tilde{\mathrm{tx}}.\mathsf{Cr}.\mathsf{c}^{\mathrm{out}}_t = \mathrm{tx}'_i.\tilde{\mathrm{tx}}.\mathsf{c}^{\mathrm{in}}$, *and (3),* $\mathrm{tx}'_i.\tilde{\mathrm{tx}}.\mathsf{c}^{\mathrm{in}} = \sum_{j=1}^{\ell_i} \mathrm{tx}'_i.\tilde{\mathrm{tx}}.\mathsf{Cr}.\mathsf{c}^{\mathrm{out}}_j$,
  where ℓ_i *is the number of output accounts in transaction* tx'_i.

We define that

$$\mathsf{trace}(\mathcal{C},\mathrm{tx}) = \begin{cases} \texttt{True} & \mathrm{tx} \textit{ is a traceable transaction on } \mathcal{C} \\ \texttt{False} & \textit{otherwise} \end{cases}$$

Non-conflicting transactions. We define what it means for two transactions to be non-conflicting. We remark that, in our system, the debited accounts need to transfer all coins they have to credited accounts; therefore, those accounts cannot be debited accounts again in any other transactions. If there is a transactions in $\mathcal{C}$ such that its input account is the same as that in the considered transaction, then the examined transaction is a conflicting transaction. Otherwise, it is a non-conflicting transaction.

Definition 3.5 (Non-conflicting Transaction)**.** *Let* $\mathcal{C} = \mathcal{C}^{\emptyset}||B_1||B_2||\ldots||B_l$ *be a blockchain where* $\mathsf{payload}(\mathcal{C}) = \{\mathrm{tx}_1,\mathrm{tx}_2,\ldots,\mathrm{tx}_n\}$ *where* $n \in \mathbb{N}$, $l \in \mathbb{N}$, *and* $n \geq l$. *We say a transaction* tx *is* not conflicting *with blockchain* $\mathcal{C}$ *if and only if predicate* $\mathsf{nonconflict}(\mathcal{C},\mathrm{tx}) = \texttt{True}$, *where*

$$\mathsf{nonconflict}(\mathcal{C},\mathrm{tx}) = \begin{cases} \texttt{True} & \forall i \in [n],\ \mathrm{tx}_i.\tilde{\mathrm{tx}}.\mathsf{a}^{\mathrm{in}} \neq \mathrm{tx}.\tilde{\mathrm{tx}}.\mathsf{a}^{\mathrm{in}} \\ \texttt{False} & \textit{otherwise} \end{cases}$$

We are now ready to describe how to validate transactions in a blockchain. Valid transactions in a blockchain should be well-formed, i.e, the transactions should be verifiable, traceable, and non-conflicting. The validation of transactions is specified in algorithm validate_tx. The algorithm takes as input an initial chain $\mathcal{C}^{\emptyset}$ and a chain $\mathcal{C}$ to be examined. If the chain $\mathcal{C}$ does not have the prefix $\mathcal{C}^{\emptyset}$ or $\mathcal{C}$ is the same as $\mathcal{C}^{\emptyset}$, then a bit b is set to `False`; otherwise, $b =$ `True`. Next, if b is `True`, then every transaction in $\mathcal{C}$ will be checked by the three predicates: verify, trace, and nonconflict. If there exists a transaction that is not verifiable, traceable, or non-conflicting; then the algorithm outputs `False`; otherwise, it returns `True`. More details can be found in Algorithm 3.

3.3 Coins in a Blockchain

In the original backbone protocol (as described in [19]), if the computing power of malicious party is more than 50% of the entire computing power, the system can be broken. Our protocol, on the other hand, is preserved in this scenario by the use of coins. In this subsection, we focus on the role of coins in our system: we will describe how to bind coins of a transaction to a block; then, how to utilize coins to define the value of a single transaction and the value of an entire set of transactions; further, we explain how a set of transactions defines the value of a subchain; after that, we show how to choose the best chain using these values other than by length.

3.3.1 Bound Transactions

As discussed in Section 1, each transaction in our protocol is bound to a certain block. We say that the transaction is bound to block B if its block section is set to be B. The definition of transaction-block binding is given as follows:

Definition 3.6 (Transaction-Block Binding)**.** *Let* tx *be a transaction and* $\hat{B}$ *be a block. We say transaction* tx *is* bound *to block* $\hat{B}$ *if it holds that* $\hat{B} = \mathrm{tx}.\tilde{\mathrm{tx}}.B$.

Algorithm 3: The *transaction validation predicate*. The input is $(\mathcal{C},\mathcal{C}^{\emptyset})$.

```
1  function validate_tx (C,C^∅)
2      b ← (C^∅ ≺ C)
3      if b = True then
4          ⟨s, X, ctr, w⟩ ← head (C)
5          s' ← H(ctr, w, G(s, X))
6          repeat
7              ⟨s, X, ctr, w⟩ ← head (C)
8              {tx_1, tx_2, ..., tx_|X|} ← X
9              for i = 1 to |X| do
10                 if verify(tx_i) = False then
11                     b ← False
12                     break
13                 if trace(C, tx_i) = False then
14                     b ← False
15                     break
16                 if nonconflict(C, tx_i) = False then
17                     b ← False
18                     break
19             C ← C[1, len(C) − 1] ;          /* Remove the head from C  */
20         until (C = C^∅) ⋁ (b = False);
21     return (b)
```

Users make use of these bindings to show support for a certain blockchain. When a user signs a new transaction and broadcasts it to the miners, the support of this transaction is tracked in a transaction appendix of that chain. The transaction appendix is defined as follows.

Definition 3.7 (Transaction Appendix)**.** *Consider a blockchain $\mathcal{C}$ and a set T of unique transactions where $T = \{\mathrm{tx}_1, \mathrm{tx}_2, \ldots, \mathrm{tx}_{|T|}\}$ and $|T|$ is the number of transactions in T. We say T is a* transaction appendix *if it holds that tx_i is bound to block* head$(\mathcal{C})$*, for all $i \in [|T|]$. For convenience, we use $\mathcal{C}.T$ to access the* transaction appendix *of $\mathcal{C}$. If a subchain includes the* head$(\mathcal{C})$ *of full chain, then it also includes the $\mathcal{C}.T$, otherwise its T is ϵ.*

We now introduce two algorithms audit and virtualize that support our use of transactions and coins in the protocol. The audit is a blockchain auditing function (Algorithm 4, below). It appends each valid transaction in X to the transaction appendix of an input chain if and only if that transaction is bound to the head of the input chain. The second algorithm, called virtualize, invokes predicate audit over a set of chains, and updates the transaction appendix of each chain in the set; see Algorithm 5 for more details.

Transaction auditing. The transaction auditing function audit is parameterized by the nonconflict predicate, the trace predicate, and the TxAuth.verify function. This algorithm accepts a blockchain $\mathcal{C}$ and transaction set X as input. It serves to audit the chain $\mathcal{C}$ with the fresh transactions found in X that are bound to the head of $\mathcal{C}$. In other words, it appends every transaction tx_i in X to the chain's transaction appendix $\mathcal{C}.T$ if it satisfies that $\mathrm{tx}_i.\tilde{\mathrm{tx}}.B = \mathsf{head}(\mathcal{C})$, tx_i does not conflict with $\mathcal{C}$ (nonconflict$(\mathcal{C}, \mathrm{tx}_i) = \mathtt{True}$), tx_i is traceable on $\mathcal{C}$ (trace$(\mathcal{C}, \mathrm{tx}_i) = \mathtt{True}$), and the authenticity of the signed transaction TxAuth.verify$(\mathrm{tx}_i) = \mathtt{True}$.

Algorithm 4: The audit function takes a chain $\mathcal{C}$ and transaction set $X = \{\mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_{|X|}\}$ as input.

Function audit $(\mathcal{C}, X)$
  $\hat{B} \leftarrow \mathsf{head}(\mathcal{C})$
  if $X \neq \emptyset$ **then**
    for $i = 1$ *to* $|X|$ **do**
      if $\mathsf{TxAuth.verify}(\mathsf{tx}_i) = \mathtt{True} \bigwedge \mathsf{tx}_i.\tilde{\mathsf{tx}}.B = \hat{B}$ **then**
        if $\mathsf{nonconflict}(\mathcal{C}, \mathsf{tx}_i) = \mathtt{True} \bigwedge \mathsf{trace}(\mathcal{C}, \mathsf{tx}_i) = \mathtt{True}$ **then**
          $\mathcal{C}.T \leftarrow \mathcal{C}.T \bigcup \{\mathsf{tx}_i\}$
  return

Virtual block extension. The virtualize is a chain extension algorithm that applies audit across a set of blockchains, denoted $\mathbb{S}$, and updating each one in turn. The virtualize algorithm artificially extends each blockchain $\mathcal{C}_i$ stored in a set $\mathbb{S} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{|\mathbb{S}|}\}$ for $1 \leq i \leq |\mathbb{S}|$ by generating or updating the transaction appendix $\mathcal{C}_i.T$ of each chain $\mathcal{C}_i$ in $\mathbb{S}$ with the transactions from the input set X. It is parameterized by the previous algorithm audit. Each candidate chain $\mathcal{C}_i$ in $\mathbb{S}$ is audited with $\mathsf{audit}(\mathcal{C}_i, X)$. This updates each chain's appendix with any new binding transactions found in X.

Algorithm 5: The virtualize function takes a set of chains $\mathbb{S} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{|\mathbb{S}|}\}$ and a transaction set X as input.

Function virtualize $(\mathbb{S}, X)$
  $\{\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathbb{S}|}\} \leftarrow \mathbb{S}$
  for $i = 1$ *to* $|\mathbb{S}|$ **do**
    $\mathsf{audit}(\mathcal{C}_i, X)$
  return

3.3.2 Coins in a Transaction, in a Block, and in a Blockchain

Transactions carry balance from account to account and coins can be transferred between users' accounts. We now introduce *transaction value* which is the total amount of coins that have been transferred from the input account to the output accounts. Briefly, the balance of the input account being spent is the value of the entire transaction. This value must be equal to the sum of the balances of the output accounts. For the sake of simplicity in our presentation, we assume all accounts are unique.

Definition 3.8 (Transaction Value)**.** *Let* $\mathsf{value}(\mathsf{tx})$ *be the transaction value for transaction* tx*. Then we have* $\mathsf{value}(\mathsf{tx}) = \sum_{k=1}^{\ell} \mathsf{tx}.\tilde{\mathsf{tx}}.\mathsf{Cr}.\mathsf{c}_k^{\mathsf{out}}$ *where* ℓ *is the number of credited accounts in* tx *and* $\mathsf{tx}.\tilde{\mathsf{tx}}.\mathsf{Cr}.\mathsf{c}_k^{\mathsf{out}}$ *refers to the amount of coins credited to account* $\mathsf{a}_k^{\mathsf{out}}$ *for* $k \in [\ell]$*.*

We extend this definition to encompass the total value of a set of transactions. The naïve approach would be to sum the transaction value of each member of the set one-by-one. This would cause a serious problem with frequently or repetitively spent coins, which would be counted again and again. An adversary playing this strategy could significantly increase the value of his preferred set. Instead, we resolve this issue by only considering the coins that are not spent in any other transactions in the set. More precisely, if there are no input accounts in the other transactions that transfer some coins, then the coins will be contributed to the value of the considered set. The formal definition of value for a set of transactions is given by the following.

Definition 3.9 (Value for a Set of Transactions)**.** *Let* $\mathsf{value}(X)$ *be the value for a set* X *of transactions where* $X = \{\mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_m\}$ *and* $m \in \mathbb{N}$*. Then we have* $\mathsf{value}(X) = \sum_{i=1}^{m} \sum_{k=1}^{\ell_i} (\mathsf{tx}_i.\tilde{\mathsf{tx}}.\mathsf{Cr}.\mathsf{c}_k^{\mathsf{out}}) \cdot \phi_{ik}$ *where* ℓ_i *is the number of credited accounts in transaction* tx_i*, and* $\phi_{ik} = 1$ *if* $\mathsf{tx}_i.\tilde{\mathsf{tx}}.\mathsf{Cr}.\mathsf{a}_k^{\mathsf{out}} \neq \mathsf{tx}_{i^*}.\tilde{\mathsf{tx}}.\mathsf{a}^{\mathsf{in}} \ \ \forall i^* \in [m]$*, otherwise* $\phi_{ik} = 0$*.*

In our system, coins are moved from accounts to others and no new coins are generated in our system. This implies that the value of any blockchain is equal to the value of the initial blockchain $\mathcal{C}^{\emptyset}$. Therefore, in order to compare two chains, we need to find the divergent position (e.g., j-th block) of the two examined chains, then

compare the two subchains obtained by truncating all blocks from the initial block to the j-th block. In other words, we need to calculate and compare the values of the two branches instead of the entire chains. Again, note that we only consider coins that are not spent when determining the value of a subchain.

We remark that the value of a subchain is essentially based on the value of its payload which is a set of transactions. Therefore, for a subchain $\mathcal{C}[j,m]$, $\mathsf{value}(\mathcal{C}[j,m]) = \mathsf{value}(\mathsf{payload}(\mathcal{C}[j,m]))$. For simplicity, we use $\mathsf{value}(\mathcal{C}[j,m])$ to indicate the value of the subchain $\mathcal{C}[j,m]$.

We further remark that, considering a subchain, coins spent over and over again are no more powerful than coins being spent once in that subchain. Please see Appendix A.1 and Figure 4 for more explanation.

3.3.3 Chain Comparison

With the definition of subchain value, we can compare the value of two chains as follows. Given two chains $\mathcal{C}_1$ of length ℓ_1 and $\mathcal{C}_2$ of length ℓ_2 with fork occurring after the j-th block. Specifically, $\mathcal{C}_1[j+1]$ and $\mathcal{C}_2[j+1]$ are not identical. We say the chain $\mathcal{C}_1$ is better than $\mathcal{C}_2$ if $\mathsf{value}(\mathcal{C}_1[j,\ell_1]) > \mathsf{value}(\mathcal{C}_2[j,\ell_2])$. If it happens that $\mathsf{value}(\mathcal{C}_1[j,\ell_1]) = \mathsf{value}(\mathcal{C}_2[j,\ell_2])$, we then say $\mathcal{C}_1$ is better than $\mathcal{C}_2$ if $H(\mathsf{head}(\mathcal{C}_1)) < H(\mathsf{head}(\mathcal{C}_2))$. We now provide a comparison algorithm, denoted compare, to implement this intuition.

Comparison. The compare algorithm is introduced to compare blockchains (see Algorithm 6). It enables the honest players to deterministically select the best chain given any two chains. The bestvalid (Algorithm 7) invokes the validate and compare functions over a set of chains, then returns the best valid blockchain.

At a high level, compare takes as input two chains $\mathcal{C}_1, \mathcal{C}_2$, and the divergent position j from their respective roots. We note that the algorithm uses a global public parameter $c_{\text{malicious}}$ which denotes the greatest number of coins controlled by the adversary, and a function max to find the maximum value of its two input values. The algorithm compare first computes the subchain values of $\mathcal{C}_1$ and $\mathcal{C}_2$ from the divergent position j. For the chain $\mathcal{C}_1$, it computes the value of $\mathcal{C}_1[j,\mathsf{len}(\mathcal{C}_1)]$, then adds the output of the max function on input $(0, \mathsf{value}(\mathcal{C}_1.T) - 1.1c_{\text{malicious}})$ where $\mathcal{C}_1.T$ is the transaction appendix of $\mathcal{C}_1$. Note that we need to subtract $1.1c_{\text{malicious}}$ coins from the appendix value in order to defend against the attack where the adversary adds his malicious coins to transaction appendix of his preferred chain. The computation of val_2 is analogous to val_1. The algorithm next compares the subchain values of each chain from the divergent position j, and then returns the chain with the greater value. In the event of a tie, the block hashes are used to make a decision by comparing the integer representation of both block's hash value. Please see Appendix A.1 and Figure 4 for an illustration of the algorithm on two chains.

Algorithm 6: The chain compare function takes $\{\mathcal{C}_1, \mathcal{C}_2, j\}$ as input.

```
Function compare (C_1, C_2, j)
    temp ← C_1
    val_1 ← value (C_1[j, len(C_1)]) + max(0, value(C_1.T) − 1.1 c_malicious)
    val_2 ← value (C_2[j, len(C_2)]) + max(0, value(C_2.T) − 1.1 c_malicious)
    if val_1 = val_2 then
        if H(head(C_1)) > H(head(C_2)) then
            temp ← C_2
    else if val_1 < val_2 then
        temp ← C_2
    return temp
```

Best valid chain. We further introduce the bestvalid algorithm that defines our strategy for selecting the best blockchain over a set of chains. The bestvalid algorithm is parametrized by the compare (Algorithm 6), validate_tx (Algorithm 3), validate_chain (Algorithm 2), and maxlen functions where maxlen takes as input a set of integers (lengths) then outputs the max length. The algorithm bestvalid takes as input a set of blockchains $\mathbb{S}$ and the initial blockchain $\mathcal{C}^{\emptyset}$. It first finds the max length m which is the length of the longest chain, then creates an array T, and checks the validity of every chain in $\mathbb{S}$. Note that the boolean array T is used to mark the chains that either have not been compared or the best chain of any pair of chains. The bestvalid

algorithm next scans every pair of chains in $\mathbb{S}$ to find the corresponding divergent position, then uses compare algorithm to compare the two considered chains from the divergent position. Once a better chain is determined, it then sets $\mathsf{T}[\text{removed}] = \texttt{False}$ where $\mathcal{C}_{\text{removed}}$ is the chain has less value so that the algorithm does not need to compare other chains with this chain later. The algorithm eventually finds the best chain in order from the newest to oldest forks. Once all chains are examined, the last $\mathcal{C}_{\text{best}}$ is the best chain among all chains in $\mathbb{S}$.

Algorithm 7: The bestvalid function takes $(\mathbb{S}, \mathcal{C}^{\emptyset})$ as input.

Function bestvalid $(\mathbb{S}, \mathcal{C}^{\emptyset})$
  Let $m = \mathsf{maxlen}(\mathsf{len}(\mathcal{C}_1), \ldots, \mathsf{len}(\mathcal{C}_{|\mathbb{S}|}))$
  Create array T where $\mathsf{T}[i] = \texttt{True}$ for all $i \in [|\mathbb{S}|]$
  for $i = 1$ *to* $|\mathbb{S}|$ **do**
    if $\mathsf{validate\_chain}(\mathcal{C}_i, \mathcal{C}^{\emptyset}) \bigwedge \mathsf{validate\_tx}(\mathcal{C}_i, \mathcal{C}^{\emptyset})$ **then**
      $\hat{\mathbb{S}} := \hat{\mathbb{S}} \bigcup \mathcal{C}_i$
  $\{\mathcal{C}_1, \ldots, \mathcal{C}_{|\hat{\mathbb{S}}|}\} \leftarrow \hat{\mathbb{S}}$
  for $j = m$ *to* 1 **do**
    for $i = 1$ *to* $|\hat{\mathbb{S}}|$ **do**
      for $k = i + 1$ *to* $|\hat{\mathbb{S}}|$ **do**
        if $\mathcal{C}_i[j] = \mathcal{C}_k[j] \neq \perp \bigwedge \mathsf{T}[i] = \mathsf{T}[k] = \texttt{True}$ **then**
          $\mathcal{C}_{\text{best}} \leftarrow \mathsf{compare}(\mathcal{C}_i, \mathcal{C}_k, j)$; /* best, removed $\in \{i, k\}$ */
          $\mathsf{T}[\text{removed}] = \texttt{False}$; /* If best $= i$ then removed $= k$, and vice versa */
  return $\mathcal{C}_{\text{best}}$

3.4 Our Protocol

As discussed in the introduction, we study a practical scenario where a cryptocurrency system has "grown up" such as the Bitcoin system, and then the adversary suddenly controls the majority of computing power. More precisely, the original system has already built a stable blockchain $\mathcal{C}^{\emptyset}$, then the adversary's computing power increases and dominates the system. At this point, all users and miners start to switch to a new mode: our modified-backbone protocol.

The notations and algorithms described in previous subsections lay the foundation for the modified-backbone protocol that we will describe in this subsection. Our modified-backbone protocol differs from that in [19], in that we rely on the miners *and the users*. Our protocol consists of two types of rounds: *miner-rounds* and *user-rounds*. For simplicity of presentation and without lost of generality, we consider an odd round as a miner-round, and an even round as a user-round; in a miner-round, only the miners are active; and in a user-round, only the users are active. Please see Figure 3 for a pictorial illustration. All players have access to BROADCAST that delivers messages to others in the next time step (round). Each player has an INPUT() tape and an OUTPUT() tape which are used to read from and write to the environment $\mathcal{Z}$, respectively. In our protocol, the players are granted (bounded) access to the random oracles $H(\cdot)$ and $G(\cdot)$. We now describe the details of our protocol.

Miners and users. The miners are the driving force of the protocol, who build blockchains; while the users are witnesses, who vote for blockchains by binding their transactions to blockchains in order to show their support. Clearly, the two roles have an interesting relationship. On one hand, we have the users who produce transactions with coins in the system; on the other hand, we have the miners who collect these transactions and include them on a blockchain.

Our protocol in miner-round is described in Algorithm 8. Here, several functions/predicates are used, including validate, virtualize, bestvalid, and pow. Now each player (i.e., miner) keeps a local storage $\hat{\mathbb{S}}$ to store a set of blockchains. Those blockchains are either received from the network via broadcast channels, or are mined by the miners themselves.

Likewise, our protocol in user-round is described in Algorithm 9. Here, function bestvalid and algorithms

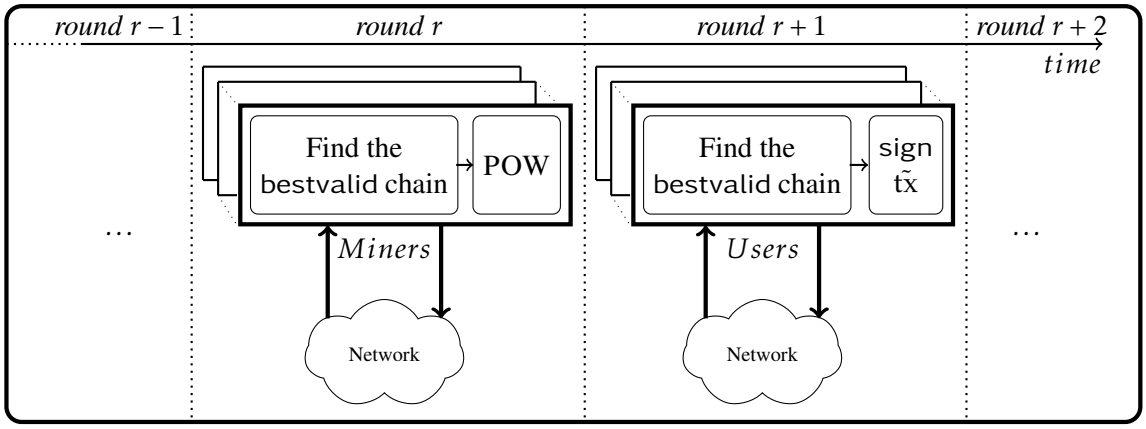

Figure 3: Round Progression in Our Protocol.

Our protocol consists of two types of rounds: *miner-round* and *user-round*. In each round, each player (miner or user) first computes its best valid chain based on its local state and the received messages from the network; then attempts to extend the chain via proof of work, or issues a transaction; and then finally broadcast its attempt or computed transactions to the network.

that related to transaction authentication scheme TxAuth are used. Now, each player (i.e, user) maintains a set of accounts.

In each miner-round or user-round, the players (both miners and users) carry out three major steps. In the first step, the miners and users first update their local views by reading the messages they have received. The miners and users update the chain set $\hat{\mathbb{S}}$ with any valid chain they find on the communication tapes RECEIVE(). Let's consider a miner. If the miner finds a transaction tx in the RECEIVE() tape, then it stores the transaction in a separate local storage called X. For every chain in $\hat{\mathbb{S}}$, the miner first checks whether the chain and all transactions on this chain are valid by using algorithms validate_chain (Algorithm 2) and validate_tx (Algorithm 3). If a chain is not valid or there exists a transaction on this chain that is not valid, then the chain will be removed from $\hat{\mathbb{S}}$. The miner then has an updated set $\mathbb{S}$ where all invalid chains are removed. Further, the miner uses the virtualize algorithm to generate or update the transaction appendix T of each chain in $\mathbb{S}$ with the transactions from the input set X. After that, the miner runs bestvalid algorithm on input $(\mathbb{S}, \mathcal{C}^{\emptyset})$ to find the best valid blockchain $\mathcal{C}$. The computation in this step of each user is similar to that of the miner except that the user does not need to verify every chain in $\hat{\mathbb{S}}$ as well as run virtualize since he is only involved in the computation to support for his best chain. Moreover, the user will find the best chain among all chains in $\hat{\mathbb{S}}$ along with his local chain.

Each miner invokes the proof of work (see Algorithm 1) with his best chain $\mathcal{C}$, the transaction appendix $\mathcal{C}.T$, and the initial chain $\mathcal{C}^{\emptyset}$ as input in the second step. On the other hand, each user receives a command from the environment $\mathcal{Z}$ in its INPUT() tape; then executes the instruction. If the command is RegisterAccount, then the user only needs to create a distinct account in the system, and then outputs it to other participants. If the command is (Credit, Argument), the user will issue a new transaction. Argument is parsed to obtain accounts $(\mathsf{a}^{\mathsf{in}}, \mathsf{a}^{\mathsf{out}})$ and the number of coins c^{in} to be debited in the new transaction. The user next binds his transaction to the head of his best chain from the earlier step, and then signs this transaction.

Finally, the players (both miners and users) share their views with other players by broadcasting their local views to the network.

Algorithm 8: Main Protocol: Miner.

$\mathcal{C} \leftarrow \mathcal{C}^{\emptyset}$; $\mathbb{S} \leftarrow \epsilon$; $round \leftarrow 0$;
while `True` **do**
  $\hat{\mathbb{S}} \leftarrow$ all chains found in RECEIVE()
  $X \leftarrow$ all transactions found in RECEIVE()
  while $\hat{\mathbb{S}} \neq \emptyset$ **do**
    $\hat{\mathcal{C}} \leftarrow$ the first element in $\hat{\mathbb{S}}$
    $\hat{\mathbb{S}} \leftarrow \hat{\mathbb{S}} - \{\hat{\mathcal{C}}\}$
    if validate_chain$(\hat{\mathcal{C}}, \mathcal{C}^{\emptyset}) \bigwedge$ validate_tx$(\hat{\mathcal{C}}, \mathcal{C}^{\emptyset})$ **then**
      $\mathbb{S} \leftarrow \mathbb{S} \cup \{\hat{\mathcal{C}}\}$
  virtualize$(\mathbb{S}, X)$
  $\mathcal{C} \leftarrow$ bestvalid$(\mathbb{S}, \mathcal{C}^{\emptyset})$
  $\mathcal{C}_{\text{new}} \leftarrow$ pow$(\mathcal{C}.T, \mathcal{C}, \mathcal{C}^{\emptyset})$
  if $\mathcal{C} \neq \mathcal{C}_{\text{new}}$ **then**
    $\mathbb{S} \leftarrow \mathbb{S} \cup \{\mathcal{C}_{\text{new}}\}$
  $round \leftarrow round + 1$; BROADCAST$(\mathcal{C})$;

Algorithm 9: Main Protocol: User.

$\mathcal{C} \leftarrow \mathcal{C}^{\emptyset}$; $round \leftarrow 0$; Account $\leftarrow \epsilon$; $command \leftarrow \epsilon$;
while `True` **do**
  $\hat{\mathbb{S}} \leftarrow$ any chain found in RECEIVE()
  $\mathcal{C} \leftarrow$ bestvalid$(\{\mathcal{C}\} \bigcup \hat{\mathbb{S}}, \mathcal{C}^{\emptyset})$
  $command \leftarrow$ INPUT()
  if $command =$ *RegisterAccount* **then**
    $(\mathsf{a}, \mathsf{sk}) \leftarrow$ TxAuth.keygen(1^{κ})
    Account $\leftarrow \{(\mathsf{a}, \mathsf{sk})\} \bigcup$ Account
    OUTPUT(a)
  else if $command =$ (*Credit, Argument*) **then**
    $(\mathsf{a}^{\text{in}}, \mathsf{a}^{\text{out}}, \mathsf{c}^{\text{in}}) \leftarrow$ Argument
    if $(\mathsf{a}^{\text{in}}, \mathsf{sk}) \in$ Account **then**
      $\tilde{\mathsf{tx}} \leftarrow \left(\mathsf{a}^{\text{in}}, \mathsf{c}^{\text{in}}, (\mathsf{a}_1^{\text{out}}, \mathsf{c}_1^{\text{out}}), \ldots, (\mathsf{a}_{\ell}^{\text{out}}, \mathsf{c}_{\ell}^{\text{out}}), \mathsf{head}(\mathcal{C})\right)$
      $\mathsf{tx} \leftarrow$ TxAuth.sign$(\mathsf{sk}, \tilde{\mathsf{tx}})$
  $round \leftarrow round + 1$;
  BROADCAST(tx); BROADCAST$(\mathcal{C})$;

3.5 Discussions

3.5.1 How to bootstrap from a mature blockchain?

As mentioned in the introduction, we are interested in bootstrapping our system from an existing transaction ledger recorded on a mature blockchain $\mathcal{C}^{\emptyset}$. A *public transaction ledger* $\mathcal{L}$ is comprised of a set of valid transactions X with an efficient membership test. The ledger $\mathcal{L}$ is one possible application that sits on top of a backbone protocol by instantiating an appropriate content validation predicate $V(\cdot)$, chain reading function $R(\cdot)$, and input contribution function $I(\cdot)$. A robust transaction ledger has two essential properties, *persistence* and *liveness*. Informally, transaction persistence describes a transaction's ability to remain in a ledger indefinitely, while liveness is the property that describes an honest user's ability to have his transactions included in a ledger.

Note that our modified-backbone protocol requires a transaction ledger for recording the transfer of coins. As described in Algorithm 9, the users generate accounts and issue transactions. Then, honest miners collect

those transactions, validate them, and append them to a transaction appendix (see Algorithm 8). Looking ahead, in Section 4, we will argue that if honest miners reach a "pure successful round", then it will inevitably lead to an "aligned round". The pure successful round guarantees that only honest miners extend a chain with a block which includes all of the valid transactions in the block's transaction appendix. As such, the set of transactions X in the new block becomes a part of the global transaction ledger $\mathcal{L}$— this achieves liveness for the transactions in X. Furthermore, the malicious miners cannot always fork the best chains before a pure successful round with a high probability, that means all of the transactions will be liveness with some probability. An aligned round guarantees that if a chain $\mathcal{C}$ is the best chain acknowledged by all honest users in a user-round, and $\mathcal{C}'$ is the best chain of an honest user in a later user-round, then $\mathcal{C} \preceq \mathcal{C}'$. In other words, if any transaction is included in a chain that is selected as the best chain at an aligned round, then it becomes a part of the global ledger indefinitely— it is persistent.

3.5.2 Enforcing honest users to stick to a single blockchain

Our protocol works because the users are frequently spending coins to support one best chain in their views. If the users do not transfer enough coins per round or they elect to support multiple chains, then we encounter some serious problems.

While we hope a user will support only one chain with his coins, doing so limits his ability to have his transactions included in the best blockchain. This is because transactions which are bound to chains that are neither extended nor included in the next block on that chain may never be included in the best blockchain. Transactions of this nature are called orphaned, and the user that issued the transaction must pick a new block to bind it to and re-broadcast the updated transaction. Users have different views of the best blockchain, and so they will sometimes have their transactions orphaned. Instead of picking just one blockchain, we expect rational users to broadcast many transactions of identical content, each with a different binding and digital signature. Following this strategy, a user increases the probability that his transactions will be included the next block. Unfortunately, when multiple chains receive the full support of the users— our consensus policy for determining the best chain breaks down. In the context of blockchains, this type of problem is known as the "nothing at stake problem"[5] because users have nothing to lose since signing transactions and binding them to blockchains are cheap and unregulated. If every user plays this strategy, then every chain would be equally supported and the common prefix property cannot be guaranteed.

In light of this, we must introduce incentives for the users to follow the protocol. Two common tactics in mechanism design come to mind: reward and punishment. Incentivizing honest behavior with rewards would work; however, our analysis assumes that there are no new coins produced in the system. Therefore, we elect to design a mechanism which punishes misbehaving parties. This will achieve the same goal without breaking the assumptions of our analysis.

We tackle this obstacle by punishing users that sign and broadcast transactions bound to different blocks using the same input account within $\Theta(\kappa)$ rounds of any other use of that input account. Each transaction that is broadcast and compared to the transaction appendix of each chain in the system; if it turns out that the same account was used as input more than one transaction within some number of rounds, then that event will be detected. This is made possible through the miners, who audit each chain during the mining rounds and broadcast the most recent set of transactions in each chain's transaction appendix. In this way, a user issuing multiple transactions from the same account will have his balance garnished by some factor. If for any reason a transaction broadcast is not confirmed or included in the next block, then the user must wait for some number of rounds before broadcasting a transaction from the same account to avoid punishment. Because of this punishment mechanism, rational users that have had their transactions abandoned will only spend their coins again after some number of rounds, lest they sacrifice utility.

[5] An instance of the economic *tragedy of the commons* fallacy, since users seemingly have nothing to lose by deviating from the protocol. This is a scenario in which each agent acts according to his own self-interest with respect to a shared and unregulated resource.

3.5.3 Increasing the honest cash flow

At the same time, we do not want users to be lazy or apathetic about the health of the system. Recall the model requires the transfer of v coins in each round. Since it is a requirement that users should spend their coins frequently, we propose to punish users that do not spend their coins often enough. This can be enforced in the same way that we punish accounts that are too active in the previous discussion. First, the activity of each account broadcasting a transaction is tracked through the miners. When a transaction refers to an account that has not been in any chain's transaction appendix for an extended number of rounds, the input account is garnished before the transaction occurs.

The upper bound and lower bound on the activity of an account frame a window of opportunity for users to preserve their wealth. This is not an unreasonable request because a user may always opt to transfer a balance to another account that he controls. Accounts with balances that are not active inside of this window will be discounted, reducing utility.

3.5.4 Considerations in Stake for Consensus

Our protocol is unique in that we use the coins and transactions of users to guide the direction of a proof of work blockchain. There are many other considerations in the literature for constructing a secure proof of stake system. At the time of writing, there is no pure proof of stake schemes that are provably secure; there are many arguments against it at the moment, many of which are described in [35]. Despite the grim outlook, there are promising solutions that fall under two categories: delegated proof of stake, where a validator set that builds the blockchain is selected based on stake, and hybrid protocols with proof of work. Many of these issues address specific pitfalls of PoS such as the nothing at stake problem, reaching consensus in the presence of prolonged forks or network segmentation, the initial distribution problem, and precomputing attacks.

We describe the nothing at stake problem above, a quick summary of the issues with forks can be quickly understood when considering a network split. There must be a deterministic way to select the best chain when the network recombines, otherwise the network may stay split and new users will be unable to determine the best chain. This is particularly difficult to define in PoS systems, so developers have called on practical Byzantine fault tolerant consensus protocols to reach consistency and agreement. And while protocols like Tendermint [24] have these properties, it suffers from some drawbacks. Namely, who will be in the validator set? New validators have to be elected to the membership resulting in an elite group. If at any point an honest super majority is lost, the protocol will halt. Moreover, the coins used for validating a blockchain are bonded and cannot be used freely. If a validator set is too big then most of the coins are immovable and sadly misses the point of currency, but if the set is too small, then an adversary with a meager sum of coins could halt consensus. Furthermore, if bonds are required for creating validators, and validators drive consensus, how is the money for bonds originally distributed in a fair way to the users? If this is done improperly, then the protocol could be broken before it begins.

Hybrid approaches have the benefit of proofs of work, which ensure that consensus may continue (albeit very slowly) in the presence of malicious stakeholders. Chains-of-Activity proposed by [7] takes this approach, where empty proof of work block headers are generated and then a subset of stakeholders selected by the block header hash sign and pick the transaction set. This makes the signing set random each block, reducing the likelihood of developing an elite group of validators. There is a real possibility of precomputing which accounts will be selected for validating, especially because the selection process uses fixed values. Moreover, because there is no deposit and therefore nothing at stake for the stakeholders, one might sign both blocks in the event of a fork.

Ethereum's Casper [38], like Tendermint, requires a deposit to participate in the consensus process: "consensus by bet". Validators become gamblers who propose blocks and bet on how likely a block is to be successful in being on the consensus branch. Block proposers are selected using an NXT [11] style random number generator. Because this is an extractor based on the previous state of the blockchain, with entropy sourced from missing block propositions, it is manipulatable; Please also see [33].

Our protocol sits in-between these proposals and those mentioned in the related work section. One, we do not tie up coins in "bonds," making our system more useful. Second, our best chain strategy is well defined. This means our system is robust against reorganization attempts, forks, and network splits because upon

resynchronization the system will still reach consensus. Because we still rely on proofs of work, our chain will still grow at least proportionally to some fraction of the honest computing power [22]. Lastly, we mitigate the initial distribution problem through bootstrapping off of an initial proof of work chain $\mathcal{C}^{\emptyset}$.

4 Security Analysis

In this section, we are going to give a complete analysis of our modified-backbone protocol. As mentioned in the introduction and in the previous section, we start with a mature cryptocurrency system where a stable blockchain $\mathcal{C}^{\emptyset}$ has been established, and then we switch to a new mode in which all players run the modified backbone protocol (see Section 3.4). As we enter the analysis of this work, it is useful to recollect our assumptions. Recall that our goal is to defend against the malicious majority of computing power; we therefore assume that only a small portion of computing power can be controlled by honest miners while the remaining could be malicious. As briefly discussed in the introduction, it is very difficult to address the issue of the malicious majority of computing power directly; we here assume that the honest users control a majority of the coins in the system, and these honest users are willing to spend a significant number of coins in each round. Before a new block is generated in a round, the users may sign some transactions in the earlier rounds. We assume that these transactions will be accumulated for the extension of the new block. For simplicity, we expect that, in each round, the honest users spend $\Theta(\kappa) \cdot c_{\text{malicious}}$ coins that originate from the blocks which were generated in $\Theta(\kappa)$ rounds earlier where κ is the security parameter and $c_{\text{malicious}}$ is the total number of malicious coins. We note that the security of our protocol will be analyzed in the (q, v)-synchronous setting (see Section 2).

4.1 Security Properties of Backbone Protocol

In [19], Garay et al introduce two fundamental security properties of the backbone protocol which call *common prefix property* and *chain quality property*, and then show how they can be used as foundation for robust public transaction ledger protocols. We then follow suit and prove slightly modified definitions that cater to the cash flow volatility v. The common prefix property is parameterized by a value $\kappa \in \mathbb{N}$. It considers an arbitrary environment and adversary in the (q, v)-synchronous setting, and it holds as long as any two honest parties' chains are different only in its most recent κ blocks.

Definition 4.1 (Common Prefix Property)**.** *The common prefix property $\mathcal{Q}_{\text{cp}}$ with parameter $\kappa \in \mathbb{N}$ states that for any two honest users P_1, P_2 with the best valid chains $\mathcal{C}_1$, $\mathcal{C}_2$ in $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$, it holds that $\mathcal{C}_1[1, \text{len}(\mathcal{C}_1) - \Theta(\kappa)] \preceq \mathcal{C}_2$ and $\mathcal{C}_2[1, \text{len}(\mathcal{C}_2) - \Theta(\kappa)] \preceq \mathcal{C}_1$.*

The second property, which we call the *chain quality property*, aims at expressing the number of honest-player contributions that are contained in a sufficiently long and continuous part of an honest player's chain. Specifically, for parameters $\ell \in \mathbb{N}$ and $\mu \in (0, 1)$, the rate of adversarial input contributions in a continuous part of an honest party's chain is bounded by μ. This is intended to capture that at any moment that an honest player looks at a sufficiently long part of its blockchain, that part will be of sufficiently "quality," i.e., the number of adversarial blocks present in that portion of the chain will be suitably bounded.

Definition 4.2 (Chain Quality Property)**.** *The chain quality property $\mathcal{Q}_{\text{cq}}$ with parameters $\mu \in \mathbb{R}$ and $\ell \in \mathbb{N}$ states that for any honest miner P with chain $\mathcal{C}$ in $\text{VIEW}^{H(\cdot)}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, q, v, z)$, it holds that for large enough ℓ consecutive blocks of $\mathcal{C}$ the ratio of adversarial blocks is at most μ.*

4.2 Preliminaries

According to the protocol we describe in the previous section, the chain selection policy of honest users and miners heavily depends on the validation and comparison of blockchains. In our setting, some of the miners are honest, every transaction generated and broadcast in the user-round by honest users will be collected by the honest miners in the next round. Thus, the existence of a certain number of honest miners guarantees that the

best chain will be broadcast to all honest users. In this way, we can be sure that the users are aware of the best chain and make good choices when binding their transactions in the next user-round.

We expect that the honest miners will generate α solutions and malicious miners will generate β solutions on average in any given round where $1 > \alpha > 0$ and $1 > \beta > 0$. We require the relationship of honest and corrupted miners to be linear, where $\alpha = \lambda\beta$, with constant $\lambda \in (0, \infty)$. This implies that we allow the malicious parties have more computing power than honest miners since λ could be in $(0, 1)$. We assume the collective honest and corrupted mining power $(\alpha + \beta) < 1$. In other words, we expect less than one proof of work solution to be produced in each round.

With our protocol in a setting with a few honest miners, armed with a shrewd user policy, and accompanied by a level set of assumptions about proof of work and communication, it is clear that many rounds will be "boring" in that nothing interesting happens. This is an intentional decision and is pivotal to our analysis of the protocol. Before moving on, we define the following two types of rounds that we are interested in.

Definition 4.3 (Aligned Round)**.** *We say a user-round r is an* aligned *round, if in this round, all honest users take the same blockchain as their best blockchains.*

Definition 4.4 (Pure Successful Round)**.** *We say a miner-round r is a* pure successful *round, if in this round, at least one honest miner finds a puzzle solution and no malicious miner is able to find any.*

We note that a pure successful round will lead to an aligned round. In next section, we prove that a pure successful round can be reached in a bounded number of rounds. These help us to show the desired security properties.

4.3 The Common Prefix Property

We now turn our attention on proving the common prefix property (Definition 4.1) for our modified-backbone protocol. The concrete statement to prove can be found in Theorem 4.11.

We start with some informal proof ideas. First, we note that a pure successful round can be reached with an overwhelming probability: though the malicious miners have more computing power, the honest miners can still be able to find a new puzzle solution before the malicious miners find too many. Then we show that, in the next round after a pure successful round, all honest users in the system will reach an agreement with an overwhelming probability. After that, we prove that after a pure successful round r, the best chain among all extended chains is better than any hidden or public chains. This chain will be chosen as the best chain by all honest users in the next user-round $r + 1$. We also show that the adversarial miners are only able to find less than κ solutions in s rounds where $s = \frac{\kappa}{\alpha}$. Putting them together, and starting from the initial blockchain $\mathcal{C}^{\emptyset}$ of the protocol execution, the system will reach a pure successful round in some limited number of rounds. Once a pure successful round is reached, an aligned round is immediately obtained after it with an overwhelming probability.

We note that when the system reaches an aligned round, all honest users take the same chain as their best blockchains. We further show that, after an aligned round, any blockchain via forking the best blockchain at a block that is generated in an earlier round (than the aligned round) will not be accepted by the honest users.

As discussed above, we can reach the conclusion that, at any round, the best blockchain will not be diverged for too long. When the best blockchains of all users are merged together, they will not be split again. Next, we thru a sequence of lemmas to demonstrate the above proof ideas in details. We remark that the parameter λ in our construction is in the range of $(0, \infty)$, and this is optimal and is much better than the range in [19]. Note that in [19], $\lambda \in [1, \infty)$ is considered, which implies that $\alpha > \beta$; that is, there, the malicious miners collectively have less computing power than honest miners. In contrast, we can show that, here, even if the adversarial miners have more than 50% of computing power, the two fundamental properties (common prefix property and chain quality property) are retained in our protocol. We prove the common prefix property by giving our first lemma showing that honest miners will often reach pure successful round with an overwhelming probability in $s = \frac{\kappa}{\alpha}$ rounds.

Lemma 4.5. *Let κ be the security parameter. Let $\alpha = \lambda\beta$, $\lambda \in (0, \infty)$, $\alpha > 0$, $\beta > 0$, $\alpha + \beta < 1$, and $s = \frac{\kappa}{\alpha}$. Let Y be the number of pure successful rounds in s rounds. We have $\Pr[Y \geq 1] \geq 1 - e^{-\Omega(\kappa)}$.*

Proof. There are $s \cdot \alpha \cdot (1-\beta)$ pure successful rounds in s rounds. By applying Chernoff bounds for $\delta \in (0,1)$, we obtain

$$\Pr[Y < (1-\delta)s\alpha(1-\beta)] \leq e^{-\delta^2 s\alpha(1-\beta)/2}$$

Let $(1-\delta)s\alpha(1-\beta) = 1$, and $s\alpha = \kappa$. This implies $\delta = 1 - \frac{1}{s\alpha(1-\beta)} = 1 - \frac{1}{\kappa(1-\beta)}$. It follows that

$$\begin{aligned}\Pr[Y < 1] &\leq e^{-(1-\frac{1}{\kappa(1-\beta)})^2\kappa(1-\beta)/2} \\ &= e^{-(\kappa(1-\beta)-2+\frac{1}{\kappa(1-\beta)})/2} \\ &\leq e^{-\Omega(\kappa)}\end{aligned}$$

Therefore, we conclude that

$$\Pr[Y \geq 1] \geq 1 - e^{-\Omega(\kappa)}$$

□

We continue the proof by showing that during two pure successful rounds, the malicious miners cannot find more than κ solutions. This also implies that there are at most $d = (1+\gamma)\frac{\kappa}{\lambda}$ branches (forks) in the main chain generated by the adversarial miners between two pure successful rounds, where $\lambda, \gamma \in (0,1)$.

Lemma 4.6. *Let κ be the security parameter. Let $\alpha = \lambda\beta$, $\lambda \in (0,\infty)$, $\alpha > 0$, $\beta > 0$, and $\alpha + \beta < 1$. Let $\gamma \in (0,1)$, and $s = \frac{\kappa}{\alpha}$. Let Z be the number of solutions that are computed by malicious miners in s rounds. We have $\Pr[Z > (1+\gamma)\frac{\kappa}{\lambda}] < e^{-\Omega(\kappa)}$.*

Proof. Similar to the Lemma 4.5, the honest miners will compute $s \cdot \alpha$ solutions on average, and the malicious miners will compute $s \cdot \beta = s \cdot \frac{\alpha}{\lambda}$ solutions on average in s rounds. By applying Chernoff bounds for $\delta \in (0,1)$, we obtain

$$\Pr[Z > (1+\delta)s \cdot \beta] \leq e^{-\delta^2 s\beta/3}$$

Let $\delta = \gamma$, and $s\alpha = \kappa$. This implies $s \cdot \beta = \frac{\kappa}{\lambda}$. Now we have

$$\begin{aligned}\Pr[Z > (1+\gamma)\frac{\kappa}{\lambda}] &\leq e^{-\gamma^2\frac{\kappa}{\lambda}/3} \\ &\leq e^{-\Omega(\kappa)}\end{aligned}$$

□

We view $s = \frac{\kappa}{\alpha}$ as a parameter. During s rounds, the system will reach a pure successful round with an overwhelming probability. Meanwhile, the malicious miners will generate no more than $(1+\gamma)\frac{\kappa}{\lambda}$ blocks with an overwhelming probability. We consider that the protocol begins from an aligned round so that the malicious miners can lead to $(1+\gamma)\frac{\kappa}{\lambda}$ divergent best chains for the honest users at most. We assume in each user-round the honest users will spend c_{spend} coins that originate from the blocks which are generated $\eta = \Theta(\kappa)$ rounds earlier. We note that there exists a chain $\mathcal{C}$ where the value of the $\mathsf{head}(\mathcal{C})$ is at least $\frac{c_{\text{spend}}}{d}$ and $d = (1+\gamma)\frac{\kappa}{\lambda} = O(\kappa)$. Since $c_{\text{spend}} = \Theta(\kappa)c_{\text{malicious}}$, we have $\frac{c_{\text{spend}}}{d} > c_{\text{malicious}}$.

Next, we will prove an essential lemma that will be helpful in our analysis. The intuition behind this lemma is that any extended chain in a pure successful round is more valuable than others that are not extended in this round. More concretely, given a chain $\mathcal{C}$ that is extended in a pure successful round r, for any other chain $\mathcal{C}'$ that is not extended in this round if a fork of $\mathcal{C}$ and $\mathcal{C}'$ occurs in $\eta = \Theta(\kappa)$ recent rounds, then $\mathcal{C}$ is better than $\mathcal{C}'$ after the pure successful round r.

Lemma 4.7. *Let κ be the security parameter. Assume that the malicious users control at most $c_{\text{malicious}}$ coins, and in each user-round the honest users transfer $c_{\text{spend}} = \Theta(\kappa) \cdot c_{\text{malicious}}$ coins that originate from the blocks which were generated in $\eta = \Theta(\kappa)$ rounds earlier. In each user-round, the honest users are at most split into*

$d = O(\kappa)$ groups (and therefore they at most follow d different best chains). Let miner-round r be a pure successful round and chain $\mathcal{C}$ be extended in round r by an honest miner. Let $\mathcal{C}'$ be a chain which is not extended in round r, and let B be the last common block of $\mathcal{C}$ and $\mathcal{C}'$, which is extended in an earlier miner-round r'.

Then we have the following: If $r - r' < \eta$, then $\mathcal{C}$ is better than $\mathcal{C}'$ after the miner-round r.

Proof. Recall that miner-round r is a pure successful round, in which chain $\mathcal{C}$ is extended and $\mathcal{C}'$ is not extended. We can conclude that the last block B in chain $\mathcal{C}'$ is extended no later than miner round $r-2$.

Let's use $\hat{\mathcal{C}}'$ to denote the public prefix (with appendix value) of chain $\mathcal{C}'$ at the end of user round $r-3$. Now in user-round $r-1$, each honest user can find a chain to support which is not worse than $\hat{\mathcal{C}}'$.

Note that in each round all honest users spend totally c_{spend} coins that originate from the blocks which were generated η rounds earlier and those honest users can at most be divided into d groups. This means there is at least one group of users who spends $\frac{c_{\text{spend}}}{d}$ coins to support their best chain (which will not be worse than $\hat{\mathcal{C}}'$). As the assumption, all of the spent coins in user round $r-1$ are from the blocks generated η rounds earlier, the spent coins in round $r-1$ will not decrease $\mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')])$.

Since all blockchains and new transactions will be sent to all miners and $\mathcal{C}$ is extended in the miner-round r, after round r (or at the beginning of user-round $r+1$) we have

$$\mathsf{value}(\mathcal{C}[j, \mathsf{len}(\mathcal{C})]) \geq \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) + \frac{c_{\text{spend}}}{d} \tag{1}$$

We also have the following:

- If chain $\mathcal{C}'$ is not a prefix of chain $\mathcal{C}$, i.e., $\mathcal{C}' \not\prec \mathcal{C}$ in miner-round r, then the adversary can add at most $c_{\text{malicious}}$ coins as the virtual block (transaction appendix) of $\mathcal{C}'$. Since $\mathsf{value}(\mathcal{C}'[j, \mathsf{len}(\mathcal{C}')]) = \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')])$ in the miner-round $r-2$ (note that $\hat{\mathcal{C}}'$ and $\mathcal{C}'$ are the same except that the rightmost block of $\mathcal{C}'$ is a fixed block, whereas the topmost block of $\hat{\mathcal{C}}'$ is a virtual block); after the miner-round r, by the Algorithm 6 we have

$$\begin{aligned} \mathsf{value}(\mathcal{C}'[j, \mathsf{len}(\mathcal{C}')]) &\leq \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) + \max(0, c_{\text{malicious}} - 1.1 c_{\text{malicious}}) \\ &= \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) \end{aligned} \tag{2}$$

- If chain $\mathcal{C}'$ is a prefix of chain $\mathcal{C}$, i.e., $\mathcal{C}' \prec \mathcal{C}$ after miner-round r, then the adversary can copy the virtual block of $\hat{\mathcal{C}}$ and append it into his chain $\mathcal{C}'$. Therefore, after the miner-round r, by the Algorithm 6 we have

$$\begin{aligned} \mathsf{value}(\mathcal{C}'[j, \mathsf{len}(\mathcal{C}')]) &\leq \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) + \max(0, \frac{c_{\text{spend}}}{d} + c_{\text{malicious}} - 1.1 c_{\text{malicious}}) \\ &\leq \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) + \frac{c_{\text{spend}}}{d} + c_{\text{malicious}} - 1.1 c_{\text{malicious}} \\ &= \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) + \frac{c_{\text{spend}}}{d} - 0.1 c_{\text{malicious}} \end{aligned} \tag{3}$$

From (1), (2) and (3), we have

$$\begin{aligned} \mathsf{value}(\mathcal{C}'[j, \mathsf{len}(\mathcal{C}')]) &\leq \mathsf{value}(\hat{\mathcal{C}}'[j, \mathsf{len}(\hat{\mathcal{C}}')]) + \frac{c_{\text{spend}}}{d} - 0.1 c_{\text{malicious}} \\ &< \mathsf{value}(\mathcal{C}[j, \mathsf{len}(\mathcal{C})]) \end{aligned}$$

We therefore conclude that $\mathcal{C}$ is better than $\mathcal{C}'$ after the miner-round r if $r - r' < \eta$. □

The following lemma shows that if the prefix has more value than another chain, then that also applies for the chain which contains the prefix. The formal is given as follows.

Lemma 4.8. *Given blockchains $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$. If $\mathcal{C}_2 \preceq \mathcal{C}_1$ and $\mathsf{value}(\mathcal{C}_2[j, \ell_2]) > \mathsf{value}(\mathcal{C}_3[j, \ell_3])$ at round r where a fork of $\mathcal{C}_2$ and $\mathcal{C}_3$ occurs after j-th block, then $\mathsf{value}(\mathcal{C}_1[j, \ell_1]) > \mathsf{value}(\mathcal{C}_3[j, \ell_3])$ at round r and any following round.*

Proof. Let $\mathsf{len}(\mathcal{C}_1) = \ell_1$, $\mathsf{len}(\mathcal{C}_2) = \ell_2$, and $\mathsf{len}(\mathcal{C}_3) = \ell_3$. We then consider the following two cases:

- If there are no coins moved from $\mathcal{C}_1[1, j]$ to $\mathcal{C}_1[\ell_2 + 1, \ell_1]$. This implies that $\mathsf{value}(\mathcal{C}_1[j, \ell_1]) = \mathsf{value}(\mathcal{C}_2[j, \ell_2])$. Since $\mathsf{value}(\mathcal{C}_2[j, \ell_2]) > \mathsf{value}(\mathcal{C}_3[j, \ell_3])$; thus, $\mathsf{value}(\mathcal{C}_1[j, \ell_1]) = \mathsf{value}(\mathcal{C}_2[j, \ell_2]) > \mathsf{value}(\mathcal{C}_3[j, \ell_3])$.
- If there are coins moved from $\mathcal{C}_1[1, j]$ to $\mathcal{C}_1[\ell_2 + 1, \ell_1]$, then $\mathsf{value}(\mathcal{C}_1[j, \ell_1]) > \mathsf{value}(\mathcal{C}_2[j, \ell_2])$. Therefore, we have $\mathsf{value}(\mathcal{C}_1[j, \ell_1]) > \mathsf{value}(\mathcal{C}_2[j, \ell_2]) > \mathsf{value}(\mathcal{C}_3[j, \ell_3])$.

For both cases, we have $\mathsf{value}(\mathcal{C}_1[j, \ell_1]) > \mathsf{value}(\mathcal{C}_3[j, \ell_3])$ at round r and any following round. This completes the proof. □

Similar to Lemma 4.7, the following lemma considers an extended chain in a pure successful round. This lemma is different from Lemma 4.7 in the case that the fork of $\mathcal{C}$ and $\mathcal{C}'$ occurs before η recent rounds. From both lemmas, we can see that no matter how far of the round where the fork happens from the current pure successful round, the extended chains will always be better than those are not extended.

Lemma 4.9. *Let κ be the security parameter. Assume that the malicious users control at most $c_{\text{malicious}}$ coins, and in each user-round the honest users transfer $c_{\text{spend}} = \Theta(\kappa) \cdot c_{\text{malicious}}$ coins that originate from the blocks which were generated in $\eta = \Theta(\kappa)$ rounds earlier. In each user-round, the honest users are at most split into $d = O(\kappa)$ groups (and therefore they at most follow d different best chains). Let miner-round r be a pure successful round and chain $\mathcal{C}$ be extended in round r by an honest miner. Let $\mathcal{C}'$ be a chain which is not extended in round r, and let B be the last common block of $\mathcal{C}$ and $\mathcal{C}'$, which is extended in an earlier miner-round r'.*

Then we have the following: If $r - r' > \eta$, then $\mathcal{C}$ is better than $\mathcal{C}'$ after the miner-round r.

Proof. Since $r - r' > \eta$, r will not be the next pure successful round after round r'. We denote r'' as the first pure successful round after round r'. Let $\bar{\mathcal{C}}$ be the best chain among all new extended chains after miner-round r''. From Lemma 4.7 we know that $\bar{\mathcal{C}}$ is the best chain after miner-round r''. That means no honest miner will work on a chain which is worse than $\bar{\mathcal{C}}$ after miner-round r''. Furthermore, no honest user will support the chain which is worse than $\bar{\mathcal{C}}$ after round r''. This implies that $\bar{\mathcal{C}}$ is the prefix of $\mathcal{C}$, i.e., $\bar{\mathcal{C}} \prec \mathcal{C}$ in miner-round r. Let the fork of $\mathcal{C}$ and $\mathcal{C}'$ occur after j-th block. From the Lemma 4.8, in round r and any following round we have

$$\mathsf{value}(\mathcal{C}[j, \mathsf{len}(\mathcal{C})]) > \mathsf{value}(\mathcal{C}[j, \mathsf{len}(\bar{\mathcal{C}})]) \tag{4}$$

From Lemma 4.7, the value of $\mathcal{C}'$ will not be increased by any honest user. After the miner-round r, we have

$$\mathsf{value}(\mathcal{C}[j, \mathsf{len}(\bar{\mathcal{C}})]) > \mathsf{value}(\mathcal{C}'[j, \mathsf{len}(\mathcal{C}')]) \tag{5}$$

From (4) and (5), it follows that

$$\mathsf{value}(\mathcal{C}[j, \mathsf{len}(\mathcal{C})]) > \mathsf{value}(\mathcal{C}[j, \mathsf{len}(\mathcal{C}')])$$

Thus, we conclude that $\mathcal{C}$ is better then $\mathcal{C}'$ after the miner-round r if $r - r' > \eta$. □

Our main lemma below states the following. Suppose there is an aligned round r', and r is the first pure successful round after r'. Then in the next user-round $r + 1$, all honest users will take the same chain as the best chain with an overwhelming probability. The usefulness of this lemma will be in showing that honest users will often reach an aligned round in $s = \frac{\kappa}{d}$ rounds in which all honest users have the same best chain. Moreover, if all honest users have the same chain $\mathcal{C}$ after the aligned round, then all other best chains of honest users in the further user-rounds will have prefix $\mathcal{C}$. This means if the best blockchains of honest users are aligned/merged to the same best chain in this user-round, then the merged part of these best blockchains will be the prefix of all honest users' best chains in later user-rounds. That is, for any user-round, all honest users will have a common-prefix with an overwhelming probability.

Lemma 4.10. *Let round r' be an aligned round, and let miner-round r be the first pure successful round after round r'. Then, we have that the next user-round $r+1$ is an aligned round with probability at least $1-e^{-\Omega(\kappa)}$ where κ is the security parameter. Furthermore, if $\mathcal{C}$ is the best chain by all honest users in user-round $r+1$ and $\mathcal{C}'$ is the best chain by an honest user in a later user-round, then we have $\mathcal{C} \preceq \mathcal{C}'$.*

Proof. From Lemma 4.5, after aligned round r', the honest miners will reach a pure successful round in $s = \frac{\kappa}{\alpha}$ rounds with probability at least $1-e^{-\Omega(\kappa)}$. At the same time, from Lemma 4.6 the malicious miners find no more than $(1+\gamma)\frac{\kappa}{\lambda}$ solutions with probability at least $1-e^{-\Omega(\kappa)}$ in s rounds. Let $\tilde{Y}$ denote the event where the malicious miners divide the honest users into $d = (1+\gamma)\frac{\kappa}{\lambda} = O(\kappa)$ groups and a pure successful round is reached in s rounds. We set $\tilde{Y} = 1$ if both events occur, and $\tilde{Y} = 0$ otherwise. We then have $\Pr[\tilde{Y}=1] \geq (1-e^{-\Omega(\kappa)})^2 \approx 1-e^{-\Omega(\kappa)}$.

Therefore, the suppositions of Lemmas 4.7 and 4.9 happen with probability at lease $1-e^{-\Omega(\kappa)}$. From Lemmas 4.7 and 4.9, there exists an extended $\mathcal{C}$ contributed by honest miners in round r which is better than any public and hidden chains. Therefore, $\mathcal{C}$ is the best chain after the pure successful round r. This implies at least one honest miner will broadcast this chain to all other users after the miner-round r, then in the next user-round $r+1$ all honest users would take $\mathcal{C}$ as their best chains. We then conclude that r is a pure successful round and the next user-round $r+1$ would be an aligned round with probability at least $1-e^{-\Omega(\kappa)}$.

In an aligned round $r+1$, all honest users have the same best chain $\mathcal{C}$. Assume to the contrary that $\mathcal{C} \npreceq \mathcal{C}'$, then there exists another chain $\mathcal{C}''$ which is the best chain of the honest user (who has $\mathcal{C}'$ as his best chain in the user-round after $r+1$) in the last aligned round $r+1$. Therefore, the honest user did not take $\mathcal{C}$ and set another chain $\mathcal{C}''$ as his best chain. This contradicts with the supposition that $r+1$ is an aligned round. This completes the proof. □

We are now ready to prove the main theorem which asserts that our protocol achieves the common-prefix property with an overwhelming probability. The theorem is formally given as follows.

Theorem 4.11. *Let κ be the security parameter. Assume $\alpha = \lambda\beta$, $\lambda \in (0,\infty)$, $\alpha > 0$, $\beta > 0$, and $\alpha+\beta < 1$. Assume that malicious users control at most $c_{\text{malicious}}$ coins, and that in each user-round honest users transfer at least $\Theta(\kappa)\cdot c_{\text{malicious}}$ coins that originate from blocks which were generated in $\Theta(\kappa)$ rounds earlier. Assume the transaction authentication scheme that used in the modified-backbone protocol is unforgeable. Let $\mathbb{S}$ be the set of the best chains of the honest parties in a round. Then the probability that $\mathbb{S}$ does not satisfy the common-prefix property with parameter κ is at most $e^{-\Omega(\kappa)}$.*

Proof. Since all miners work on the same initial blockchain $\mathcal{C}^{\emptyset}$ at the very beginning, the protocol starts from an aligned round. Let r' be the aligned round. Let $s = \frac{r}{\alpha}$. We suppose r is the first pure successful round after round r'.

From Lemma 4.5, we have $\Pr[Y \geq 1] \geq 1-e^{-\Omega(\kappa)}$ where Y is the number of pure successful round in s rounds. This implies the probability the honest miners reach a pure successful round is no less than $1-e^{-\Omega(\kappa)}$.

From Lemma 4.6, we have $\Pr[Z > (1+\gamma)\frac{\kappa}{\lambda}] < e^{-\Omega(\kappa)}$ where Z denotes the number of solutions that are computed by malicious miners in s rounds. This implies the probability of the number of solutions that are computed by malicious miners in s rounds is greater than $(1+\gamma)\frac{\kappa}{\lambda}$ is at most $e^{-\Omega(\kappa)}$.

Since the malicious miners will spend at least 1 block in order to fork a branch of the chain, the total number of the forked branches in s rounds is less than $(1+\gamma)\frac{\kappa}{\lambda}$. From Lemma 4.10, we have that round $r+1$ is an aligned round with probability at least $1-e^{-\Omega(\kappa)}$.

Therefore, if there are two chains in $\mathbb{S}$, the divergent parts of the two chains are generated by malicious miners in s rounds. Then we have $\Pr[Z > (1+\gamma)\frac{\kappa}{\lambda}] < e^{-\Omega(\kappa)}$. This completes the proof of the common-prefix property. □

Remark 4.12. *Recall that our protocol satisfies the common-prefix property even if $\lambda \in (0,1)$ which GKL model does not. Moreover, in our system, the honest users should use at least $\Theta(\kappa)\cdot c_{\text{malicious}}$ coins totally in each round where κ is the security parameter and $c_{\text{malicious}}$ indicates the total number of coins controlled by the adversary.*

This concludes our proof of the common prefix property $\mathcal{Q}_{\text{cp}}$. In next subsection, we will shift our attention to proving the chain quality property $\mathcal{Q}_{\text{cq}}$, a measure of the honest miners' contributions to a blockchain over a period of time.

4.4 The Chain Quality Property

The chain-quality property (Definition 4.2) ensures that the rate of adversarial input contributions in a continuous part of an honest party's chain is bounded. From Lemma 4.10, in a pure successful round, the honest miners find at least one solution, and no malicious miners find any. We then find the lower bound of the number of blocks produced by the honest miners. We further show that given a pure successful round, the number of blocks produced by the adversarial miners is bound by their computing power. Finally, we demonstrate that the ratio of malicious blocks in an honest player's blockchain does not exceed a suitable upper bound in a sufficient number of rounds with an overwhelming probability.

Theorem 4.13. *Let κ be the security parameter. Assume $\alpha = \lambda\beta$, $\lambda \in (0, \infty)$, $\alpha > 0$, $\beta > 0$, and $\alpha + \beta < 1$ and $\psi = 1 - \beta$. Assume that malicious users control at most $c_{\text{malicious}}$ coins, and that in each user-round honest users transfer at least $\Theta(\kappa) \cdot c_{\text{malicious}}$ coins from blocks which were generated in $\Theta(\kappa)$ rounds earlier. Assume the transaction authentication scheme that used in the modified-backbone protocol is unforgeable.*

Let $\mathcal{C}$ be a blockchain for an honest user. Consider any ℓ consecutive blocks in $\mathcal{C}[i, i+\ell]$ which are generated in s rounds. The probability that the adversary has contributed no more than $\frac{1}{\frac{\psi}{4}\lambda+1}\ell$ of these blocks is at least $\text{erf}(\sqrt{\frac{t}{2}})$, $t = \Theta(s)$.

Remark 4.14. *Before going to the proof details, we remark that, in s rounds, miners generate at most $(\beta+\alpha)s$ blocks. That means $\ell \leq (\beta+\alpha)s$, and we can further derive that $s \geq \frac{\ell}{\beta+\alpha}$. Since now we have $t \geq \Theta(\ell)$ and $\text{erf}(\sqrt{\frac{t}{2}}) \approx 1$, from the above theorem, we can show that the adversary cannot contribute more than $\frac{1}{\frac{\psi}{4}\lambda+1}\ell$ out of ℓ blocks, except negligible probability.*

Proof. Suppose that block $\mathcal{C}[i]$ is generated in round r, and block $\mathcal{C}[i+\ell]$ is generated in round r' where $s = r' - r + 1$. From Lemma 4.10, each pure successful round will contribute a block to the best blockchain with the probability close to 1. For simplicity, we assume that a pure successful round will contribute a block to the best chain. The number of blocks contributed by malicious miners in the considered chain will not be greater than the total number of blocks he can generate. Let X denote the number of pure successful rounds and Z denote the number of blocks generated by malicious miners in s rounds.

For each round, the probability that it is a pure successful round is $\alpha \cdot \psi$. The distribution of random variable X will follow Binomial distribution $f(k, s, \alpha\psi)$. We assume that s is big enough, i.e., $s\alpha\psi > 5$, then X can be approximated to a normal distribution $\mathcal{N}(s\alpha\psi, s\alpha\psi(1-\alpha\psi))$. Similarly, Z will follow a normal distribution $\mathcal{N}(s\beta, s\beta(1-\beta))$.

From the property of normal distribution, we then have

$$\Pr[X < s\alpha\psi - t_1\sqrt{s\alpha\psi(1-\alpha\psi)}] < \frac{1}{2}[1 - \text{erf}(\frac{t_1}{\sqrt{2}})]$$

$$\Pr[Z > s\beta + t_2\sqrt{s\beta\psi}] < \frac{1}{2}[1 - \text{erf}(\frac{t_2}{\sqrt{2}})]$$

Let $t_1 = \frac{1}{2}\sqrt{\frac{s\alpha\psi}{1-\alpha\psi}}$, $t_2 = \sqrt{\frac{s\beta}{1-\beta}}$, $p = \max(\frac{1}{2}[1 - \text{erf}(\frac{t_1}{\sqrt{2}})], \frac{1}{2}[1 - \text{erf}(\frac{t_2}{\sqrt{2}})])$. It follows that

$$\Pr[X < \frac{1}{2}s\alpha\psi] < p$$

$$\Pr[Z > 2s\beta] < p$$

We consider the situation that $X > \frac{1}{2}s\alpha\psi$ and $Z < 2s\beta$ occur at the same time. The probability is no less than $(1-p)^2 > 1-2p$. That is

$$\begin{aligned}\frac{Z}{X+Z} &< \frac{2s\beta}{\frac{1}{2}s\alpha\psi + 2s\beta} \\ &= \frac{4\beta}{\alpha\psi + 4\beta} \\ &= \frac{1}{\frac{\psi}{4}\lambda + 1}\end{aligned}$$

We then have

$$\Pr\left[Z < \frac{1}{\frac{\psi}{4}\lambda+1}\cdot(X+Z)\right] \geq (1-p)^2 > 1-2p = \mathrm{erf}(\sqrt{\frac{t}{2}})$$

This probability is based on the number of rounds, and it is easy to see that the probability also holds if it is based on the number of blocks. Therefore, the probability that the adversary has contributed no more than $\frac{1}{\frac{\psi}{4}\lambda+1}\ell$ of these blocks is at least $\mathrm{erf}(\sqrt{\frac{t}{2}})$ where $t = \Theta(s)$. This completes the proof of the chain quality property. □

References

[1] R3 connects 11 banks to distributed ledger using Ethereum and Microsoft Azure. International Business Times, 2016. http://ibt.uk/A6SNI.

[2] J. Alwen and J. Blocki. Efficiently computing data-independent memory-hard functions. Cryptology ePrint Archive, Report 2016/115, 2016. http://eprint.iacr.org/2016/115.

[3] J. Alwen, B. Chen, C. Kamath, V. Kolmogorov, K. Pietrzak, and S. Tessaro. On the complexity of scrypt and proofs of space in the parallel random oracle model. To appear in *EUROCRYPT 2016*, 2016. http://eprint.iacr.org/2016/100.

[4] J. Alwen and V. Serbinenko. High parallel complexity graphs and memory-hard functions. In R. A. Servedio and R. Rubinfeld, editors, *47th ACM STOC*, pages 595–603. ACM Press, June 2015.

[5] M. Andrychowicz and S. Dziembowski. PoW-based distributed cryptography with no trusted setup. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 379–399. Springer, Heidelberg, Aug. 2015.

[6] A. Back. Hashcash — A denial of service counter-measure. 2002. http://hashcash.org/papers/hashcash.pdf.

[7] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld. Proof of activity: Extending Bitcoin's proof of work via proof of stake. Cryptology ePrint Archive, Report 2014/452, 2014. http://eprint.iacr.org/2014/452.

[8] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2: the memory-hard function for password hashing and other applications. 2015. Specification available https://www.cryptolux.org/index.php/Argon2.

[9] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[10] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, and A. T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

[11] N. Community. Nxt whitepaper, 2014. https://www.dropbox.com/s/cbuwrorf672c0yy/NxtWhitepaper_v122_rev4.pdf.

[12] H. Corrigan-Gibbs, D. Boneh, and S. Schechter. Balloon hashing: Provably space-hard hash functions with data-independent access patterns. Cryptology ePrint Archive, Report 2016/027, 2016. http://eprint.iacr.org/2016/027.

[13] CryptoManiac. Proof of stake. *NovaCoin wiki*, 2014. https://github.com/novacoin-project/novacoin/wiki/Proof-of-stake.

[14] J. R. Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[15] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, Aug. 1993.

[16] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, Aug. 2015.

[17] I. Eyal. The miner's dilemma. In *2015 IEEE Symposium on Security and Privacy*, pages 89–103. IEEE Computer Society Press, May 2015.

[18] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, Mar. 2014.

[19] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, Apr. 2015.

[20] D. Goodin. Bitcoin security guarantee shattered by anonymous miner with 51% network power. http://arstechnica.com/, 2014.

[21] J. Katz, A. Miller, and E. Shi. Pseudonymous broadcast and secure computation from cryptographic puzzles. Cryptology ePrint Archive, Report 2014/857, 2014. http://eprint.iacr.org/2014/857.

[22] A. Kiayias and G. Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015. http://eprint.iacr.org/2015/1019.

[23] S. King and S. Nadal. PPCoin: Peer-to-peer crypto-currency with proof-of-stake. 2012. 2014. http://www.peercoin.net/assets/paper/peercoin-paper.pdf.

[24] J. Kwon. Tendermint: Consensus without mining, 2014. http://tendermint.com/docs/tendermint.pdf.

[25] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490. IEEE Computer Society Press, May 2014.

[26] A. Miller, A. E. Kosba, J. Katz, and E. Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 680–691. ACM Press, Oct. 2015.

[27] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf.

[28] A. Narayanan, J. Bonneau, , E. W. Felten, A. Miller, and S. Goldfeder. Bitcoin and cryptocurrency technology, 2015. https://www.coursera.org/learn/cryptocurrency.

[29] National Science Foundation. NSF awards $74.5 million to support interdisciplinary cybersecurity research, 2015. http://www.nsf.gov/news/news_summ.jsp?cntn_id=136481.

[30] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. Cryptology ePrint Archive, Report 2015/796, 2015. http://eprint.iacr.org/2015/796.

[31] S. Park, K. Pietrzak, A. Kwon, J. Alwen, G. Fuchsbauer, and P. Gaži. Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. http://eprint.iacr.org/2015/528.

[32] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Cryptology ePrint Archive, Report 2016/454*, 2016. https://eprint.iacr.org/2016/454.

[33] C. Pierrot and B. Wesolowski. Malleability of the blockchain's entropy. Cryptology ePrint Archive, Report 2016/370, 2016. http://eprint.iacr.org/2016/370.

[34] D. Pike, P. Nosker, D. Boehm, D. Grisham, S. Woods, and J. Marston. Post white paper.

[35] A. Poelstra. On stake and consensus. 2015. https://download.wpsoftware.net/bitcoin/pos.pdf.

[36] M. Rosenfeld. Meni's implementation. *Bitcoin wiki URL: https://en.bitcoin.it/wiki*, 2012.

[37] L. Torvalds. Linux foundation unites industry leaders to advance blockchain technology. Technical report, The Linux Foundation, 2016.

[38] Vitalik Buterin. Understanding serenity, part 2: Casper, 2015. https://blog.ethereum.org/2015/12/28/understanding-serenity-part-2-casper/.

A Supportting Material for Our Construction

A.1 Transaction flow

We here illustrate the general "flow" of coins through the transaction graphs nested inside the payload of a few blocks below. Each block is delimited by two adjacent vertical lines. In the two illustrations below, any account's balance is easily validated and traceable to a coin base transaction in the initial state $\mathcal{C}^{\emptyset}$. The light dashed edge(s) indicates that a transaction output from a previous block is being consumed as an input to a transaction in a new block, which also specifies new transaction output(s) (solid dark edges), the original input is considered to be spent as soon as it is consumed in the best chain. A chain's transaction appendix is expressed as a square node with dotted lines that indicate their value. In this way, each leaf node has a spendable balance which is equal to the weight of the incoming edge, and the value of a given block is equal to the flow of coins entering that block (exclude the coins flowing past it). Below, we present a "toy example" to show how blocks and chains may be valued in the event of a fork. In this particular case, the chains fork at the second block and we denote the top chain as $\mathcal{C}_1$ and the bottom as $\mathcal{C}_2$. Comparing these two chains, the chains diverge at height 3 so for each chain we calculate the subchain value from the fork. In this example, we assume that $c_{\text{malicious}}$ is 10. In the case of $\mathcal{C}_1$, we have the value of the subchain from the fork is $\mathsf{value}(\mathcal{C}_1[3,\mathsf{len}(\mathcal{C}_1)]) + \max\Big(0, \mathcal{C}_1.T - 1.1 \cdot c_{\text{malicious}}\Big) = 38$. In the case of $\mathcal{C}_2$, the subchain value is $\mathsf{value}(\mathcal{C}_2[3,\mathsf{len}(\mathcal{C}_2)]) + \max\Big(0, \mathcal{C}_2.T - 1.1 \cdot c_{\text{malicious}}\Big) = 10$. Since the subchain value of $\mathcal{C}_1$ is greater than that of $\mathcal{C}_2$, the chain $\mathcal{C}_1$ would be the better chain between the two chains.

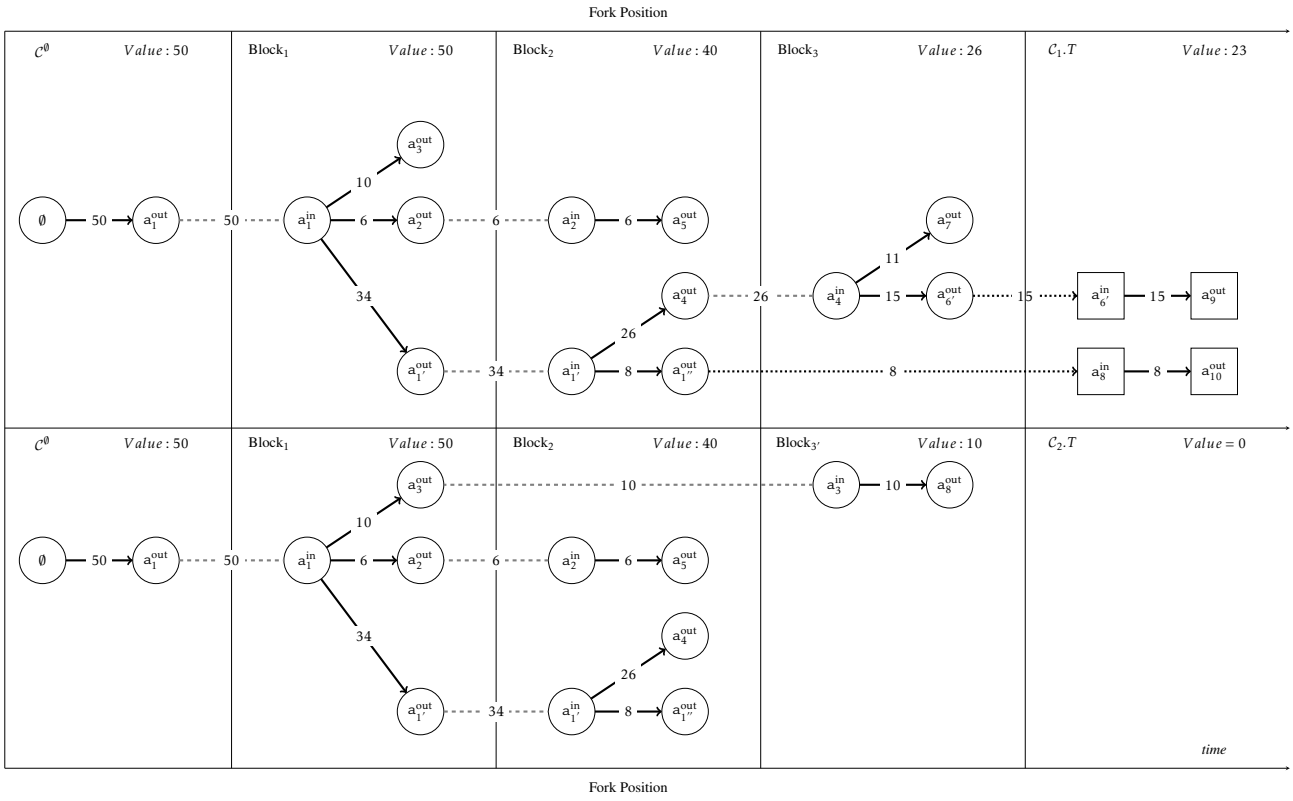

Figure 4: The flow of transactions is a weighted directed acyclic graph, and this transaction graph exists *within* the payload of a blockchain. The vertical lines represent blocks, the nodes represent accounts, and the edges show the flow of coins. The initial chain $\mathcal{C}^{\emptyset}$ contains a single coinbase transactions $\emptyset$ that is spent across three blocks. The value of each block is calculated following definition 3.9. In this case, the top chain is the best chain since it has a large coin flow entering block 3 from the fork at block 2. This chain gains strength as more users sign it with transactions, which are added to the appendix (squares).

A.2 Coinbase transaction

In Figure 5, we present the format of a *coinbase transaction*. Coinbase transactions are not signed. Note that these unsigned transactions may be included in a blockchain via proof of work (as in Bitcoin).

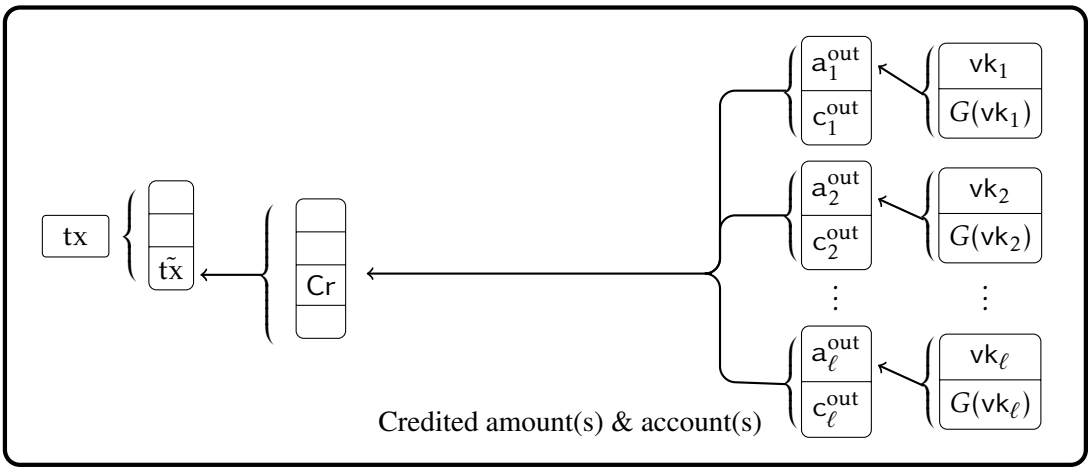

Figure 5: Coinbase Transaction Data-structure Hierarchy.