Multi-mode Cryptocurrency Systems

Tuyet Duong
Virginia Commonwealth University
duongtt3@vcu.edu

Alexander Chepurnoy
Ergo Platform and IOHK Research
alex.chepurnoy@iohk.io

Hong-Sheng Zhou
Virginia Commonwealth University
hszhou@vcu.edu

ABSTRACT

In the past years, the security of Bitcoin-like protocols has been intensively studied. However, previous investigations are mainly focused on the *single-mode* version of Bitcoin protocol, where the protocol is running among full nodes (miners). In this paper, we initiate the study of *multi-mode* cryptocurrency protocols. We generalize the recent framework by Garay et al. (Eurocrypt 2015) with new security definitions that capture the security of realistic cryptocurrency systems (e.g. Bitcoin with full and lightweight nodes). We provide the first rigorous security model for addressing the "blockchain bloat" issue. As an immediate application of our new framework, we analyze the security of existing blockchain pruning proposals for Bitcoin aiming to improve the storage efficiency of network nodes by pruning unnecessary information from the ledger.

ACM Reference Format:
Tuyet Duong, Alexander Chepurnoy, and Hong-Sheng Zhou. 2018. Multi-mode Cryptocurrency Systems. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Bitcoin [13] is a decentralized digital cash system built on top of a distributed, append-only public transaction ledger. The public ledger is maintained by a network of nodes via the proof-of-work mechanism, and only valid transactions are supposed to be recorded in the ledger. The techniques behind have proven to be very promising. Besides cryptocurrencies, many interesting applications, such as decentralized crowdfunding and decentralized autonomous organization, come to the reality.

Bitcoin-like cryptocurrencies are often multi-mode systems: some network nodes are running in "full mode", and each full-mode node records and checks the complete history of transactions, and is capable of verifying the integrity of history and the validity of new transactions by itself; while the remaining nodes may run in certain "light modes", and they might store only a partial history of transactions. More concretely, Bitcoin has multiple modes by design: in its whitepaper by Nakamoto [13], a light mode called "SPV (Simple Payment Verification)" mode has been introduced[1]. Later, in the version 0.11 of Bitcoin Core [1], another light mode called "pruning" mode has been introduced. Similarly, Ethereum [22] is also a multi-mode system. For example, Parity clients [20] for Ethereum blockchain can be run in a light mode called "WarpSync" mode [21].

The advantage of having multiple modes is very clear. First, it partially addresses the very urgent issue of "blockchain bloat": when more transactions are made, the blockchain has more data to record, and if it grows too large, it becomes difficult to download or store it; as a result, the scope of Bitcoin blockchain is very limited. Compressing the transaction history in a light mode will directly reduce the effort of storing blockchain transactions. Second, having multiple modes allows more players, especially those with limited storage, to join the system. This apparently will make multi-mode Bitcoin like cryptocurrencies more popular.

Rigorous security analysis of Bitcoin protocols has been recently developed. Garay et al. [5] proposed the first cryptographic framework and investigated the security of the Bitcoin backbone protocol. More concretely, Garay et al. investigated a single mode version of Bitcoin protocol where all the nodes are running in a full-mode, and they showed that several important security properties can be achieved under the assumption that the majority of the full-mode nodes are honest. This result has been further extended; see [6, 16]. However, the security of Bitcoin, as a multi-mode system, has not been investigated, to the best of our knowledge. This motivates the following question:

> *What security properties can the current multi-mode Bitcoin system achieve?*

1.1 Our contributions

We answer the above question by making the following contributions.

Formal treatment for the multi-mode Bitcoin system. We are the first to propose a rigorous model and analysis for multi-mode cryptocurrency systems. More specifically, we first define security properties for lightweight nodes (running in light modes) in Bitcoin system. Note that, in the previous efforts [5, 16], all transactions are faithfully recorded, and certain security properties (e.g, persistence and liveness) can be naturally defined. Now in a light mode, lots of information of the transactions will be eliminated. Unlikely, the previous security definitions will work in all light modes. We begin by introducing a generalized notion of ledger called *snapshot*. Based on the notion of snapshot, we define the relaxed security definitions: *snapshot persistence* and *snapshot liveness*.

Once the security definitions for light modes have been defined, we need to further study the security for systems with multiple modes. It is possible that nodes in different modes may not be able to work together. If that is the case, the entire system may be insecure. Therefore, a new security definition, namely, *multi-mode soundness property*, is needed to ensure the nodes in different

[1] Please see Section 8 of the whitepaper [13] for details.

Conference'17, July 2017, Washington, DC, USA
© 2018 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

modes are compatible. Intuitively, for each new transaction, nodes in the full mode and nodes in the light modes should make the same decision on whether to accept the new transaction or reject it.

Applications. As an immediate application of our analysis framework, we for the first time, provide security analysis for the two-mode version of Bitcoin (and also Ethereum) which consists of full-mode nodes and prune-mode nodes. See the pruning proposals from Bitcoin/Ethereum community; there, network nodes are allowed to be in the prune-mode, where, instead of storing the complete history of transactions, each prune-mode node keeps a succinct archive for the transaction history. These proposals have been widely adopted in cryptocurrency community. However, their security has not been investigated yet. We show that, under assumption that if each honest player in prune-mode faithfully keeps its succinct archive, then the Bitcoin/Ethereum pruning proposals can achieve the relaxed security properties. Note that the relaxed security properties are sufficient for typical blockchain applications such as cryptocurrency. More details can be found in Sections 5,6, and 7.

1.2 Related work

Cryptocurrency client modes. Multi-mode paradigm was introduced in the original Bitcoin whitepaper [13]: it describes a "full" mode node design, and also introduces a lighter mode called "SPV (Simple Payment Verification)" mode (see Section 8 of the whitepaper for details). A full node is checking everything in the blockchain: proofs of work, correctness of inter-block pointers, signatures and semantic rules for all the transactions, forking rules etc. Since version 0.11 of Bitcoin Core [1], a node can be run in another lighter mode called "pruning" mode. A node in this mode is downloading and processing all the blocks, and then leaves only a fixed-sized suffix of the blockchain assuming that some other nodes in the network store the prefix. To relax this assumption (and also to support better SPV clients), Ethereum [22] has representation of transaction validation state being fixed by the protocol with an authenticating digest to be included into a block. Some implementations of the protocol are using the digest to obtain a verifiable state snapshot from peers without building it by processing blocks from genesis. WarpSync [15] mode in Parity [20] is the example of such an implementation, it could be seen as the development of the pruning mode in Bitcoin, as a node running in this mode is not only storing a suffix of the blockchain, but also processing only fixed number of last blocks after checking block headers and downloading authenticated validation state before the full blocks. The underlying assumption is that there are network nodes storing authenticated validation states buried deep enough in the history. The scheme lacks rigorous security analysis.

Blockchain security analysis. The security of Bitcoin protocol has been extensively analyzed in both rational ([3, 4, 8, 14, 17, 18]) and cryptographic ([5, 9, 10, 16, 19]) settings. Three crucial security properties, namely *common prefix* ([5], [16]), *chain growth* ([9]) and *chain quality* ([5]) have been considered for blockchain protocols. For our study we adopt the stronger variant of the common prefix property by Pass et al. [16] together with the chain quality and chain growth from [5, 9].

Organization: We introduce the model in Section 2. Then we introduce the proof of work blockchain in Section 3. Our major contributions can be found in Section 4 for generalized ledger and the security definitions, and in Section 5, 6, and 7, for the analysis of full mode, prune mode, and multi-mode, respectively. Additional materials for those sections can also be found in appendices.

2 MODEL

Based on the previous modeling efforts [5, 16], we consider a framework with *stateful* miners, in the sense that they are required to store blockchain information locally; in this way, only blocks, not the entire blockchains, appear in the broadcast channels. We also consider different roles among the players: some players will record the complete history of transactions, and some may record a compressed version of the history. Our modeling choice is consistent with the reality.

2.1 The model of protocol execution

Network communication. We consider a standard multi-player communication setting with the relaxation that *(i)* all nodes are connected via single- or multi-hop communication channels; *(ii)* these communication channels are reliable but not authenticated, and the adversary may "spoof" the source of a message in a communication channel and impersonate the (honest) sender of the message; and *(iii)* the messages between honest players can be delayed by at most Δ number of execution rounds for some $\Delta \in \mathbb{R}^+$, and the adversary is not allowed to stop the messages from being delivered.

We use Broadcast to denote the atomic unauthenticated broadcast functionality in this asynchronous networks with Δ-bounded delay; here an adversarial sender may abuse Broadcast by sending inconsistent messages to different honest miners to confuse them. 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.

The execution of proof-of-work blockchain protocol. We consider the execution of a multi-party protocol Π among a set $\mathcal{P}$ of miners that is directed by an environment $\mathcal{Z}(1^\kappa)$, where κ is a security parameter. Each player will have the following phases.

All miners are created by the environment $\mathcal{Z}$. Once created, the miner $P \in \mathcal{P}$ becomes active with initial state which consists of the genesis block and mode information. Here the mode information specifies which role the miner P will play in the multi-party protocol.

Preparation phase. Any active miner P, before participating in the mining process, will obtain additional information from the system based on its initial state. More concretely, if the miner P is in the full mode, then P will obtain the entire blockchain information from the system, and store the blockchain information in its full-mode storage. If P is in a light mode, then it will obtain a compressed version of the blockchain information, and then store the compressed blockchain in its light mode storage.

Execution phase. Any active miner P can join the mining process after the preparation phase. The mining process consists of multiple rounds. In each round, the environment $\mathcal{Z}$ provides inputs for all miners and receives outputs from these miners, and the miners communicate with each other. More concretely, in each round, each honest miner receives an input from the environment $\mathcal{Z}$, and potentially receives incoming network messages (delivered by the adversary $\mathcal{A}$), and then updates its local storage; then based on the stored information, it carries out some mining operations; in the case that a new block is generated, the miner sends out the new block via BROADCAST which will be guaranteed to be delivered to all miners in the beginning of the next round. Note that, at any point of the execution, the environment $\mathcal{Z}$ can communicate with the adversary $\mathcal{A}$ or access the local information of the miners.

For simplicity, we consider the static computing power setting (where the total amount of computing power invested to the protocol will not change over time). We further assume all miners have the same amount of computing power, and there are n number of active miners. Moreover, we assume players remain in the same mode during the execution (i.e., no mode changing). Note that, in each execution round, all miners have access to a random oracle $hash(\cdot)$.

We consider adaptive adversaries who are allowed to take control of protocol players on the fly. At any point of the execution, $\mathcal{Z}$ can send message $(\texttt{Corrupt}, P)$ to adversary $\mathcal{A}$. From that point, $\mathcal{A}$ has access to the party's local state and controls P.

Let $\{\textsc{View}^{P}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*}$ denote the random variable ensemble describing the view of a miner P 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 security parameter κ. For simplicity, the parameters κ and z are often dropped if the context is clear, and we describe the ensemble by $\textsc{View}^{P}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$. The concatenation of the view of all miners $\langle \textsc{View}^{P}_{\Pi,\mathcal{A},\mathcal{Z}} \rangle_{P\in\mathcal{P}}$ is denoted by $\textsc{View}_{\Pi,\mathcal{A},\mathcal{Z}}$.

3 PROOF-OF-WORK BLOCKCHAIN

In this section, we provide definitions for a proof-of-work blockchain. The definitions are similar to previous works [5, 16]. Developing their approach, we put details into block and blockchain definitions which allow us to build different modes of operation for a protocol participant.

3.1 Building block: Authenticated data structure

An authenticated data structure such as Merkle tree [11] is simply a binary tree over inputs $x_1, \ldots, x_j$, such as the inputs are placed in the leaves of the tree (if j is not a power of two, we add null-elements up to a closest power of two), and the value of each internal node then is the hash of values of its two children. It is easy to see that the tree is of length $\log(j)$ and has a single authenticating digest at the top. A Merkle tree is collision-resistant. By using a Merkle tree it is possible to prove membership of any element in a set by providing logarithmic in a size of the set number of hashes. We are interested in collision-resistance only, and we define a generalized notion of an *authenticated data structure* over a data structure.

Definition 3.1. An authenticated data structure Σ_{au} is a pair of deterministic polynomial-time algorithms $(\mathsf{Root}, \mathsf{CheckRoot})$, with security parameter κ, such that:

- The digest generation algorithm Root takes a data structure x and outputs a digest $\tau \in \{0,1\}^{\kappa}$. We write this as $\tau := \mathsf{Root}(x)$.
- The digest verification algorithm CheckRoot takes a data structure $x \in \{0,1\}^*$, and a digest $\tau \in \{0,1\}^{\kappa}$. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We write this as $b := \mathsf{CheckRoot}(x, \tau)$.

We require the authenticated data structure $\Sigma_{\mathsf{au}} = (\mathsf{Root}, \mathsf{CheckRoot})$ to be collision-resistant.

Definition 3.2. An authenticated data structure $\Sigma_{\mathsf{au}} = (\mathsf{Root}, \mathsf{CheckRoot})$ is collision-resistant if for all PPT adversaires $\mathcal{A}$, it holds that

$$\Pr[\texttt{Auth-coll}_{\mathcal{A},\Sigma_{\mathsf{au}}}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$$

where $\texttt{Auth-coll}_{\mathcal{A},\Sigma_{\mathsf{au}}}(\kappa)$ is an collision-finding experiment defined as follows.

- The adversary $\mathcal{A}$ is given input 1^{κ}, and outputs x, x'.
- The output of the experiment is defined to be 1 if and only if $x \neq x'$, $\mathsf{Root}(x) = \mathsf{Root}(x') = \tau$, and $\mathsf{CheckRoot}(x, \tau) = 1$ and $\mathsf{CheckRoot}(x', \tau) = 1$. In such a case we say that $\mathcal{A}$ has found a collision.

3.2 A block and a blockchain

To capture the real-world Bitcoin system, we define a block as $\mathsf{B}_j = \langle \mathsf{head}_j, x_j \rangle$, where head_j is a *header* of the block, and x_j is a block *payload.* The header is defined as $\mathsf{head}_j = \langle h_j, \tau_j, w_j \rangle$, where $h_j = hash(\mathsf{head}_{j-1})$ and head_{j-1} is the header of the previous block is a link to a previous block, τ_j denotes the one-way digest of x_j (in the Bitcoin protocol it is generated by an authenticated data structure π_{au}, see Definition 3.1 further), and w_j is a proof-of-work puzzle solution. The header should satisfy the inequality $hash(\mathsf{head}_j) < \mathrm{T}$, where T is the proof-of-work target setting hardness of a proof-of-work puzzle solution.

A *blockchain* is a sequence of ordered and linked blocks $\mathsf{B}_{\emptyset}, \mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_{\ell}$, so for any block B_j, its header contains correct link $h_j = hash(h_{j-1})$. We denote the blockchain as $\mathsf{C} = \mathsf{B}_{\emptyset}||\mathsf{B}_1||\mathsf{B}_2||\ldots||\mathsf{B}_{\ell}$, where operation "$||$" indicates the concatenation between any two blocks. Here, we denote $\mathsf{B}_{\emptyset}$ as the genesis block.

Blockchain payload and payload validation predicate. A *blockchain payload* is concatenation of blockchain block payloads. In details, for blockchain C, blockchain payload is $x_{\mathsf{C}} = \langle x_1, \ldots, x_{\ell} \rangle$. The validation of the blockchain payload depends on the concrete applications on top of the blockchain protocol. In our blockchain protocol, participants are validating the puzzle solutions via checking the hash inequality, the links between blocks via the hash chain, and the payloads via a deterministic predicate $\mathbf{V}(\cdot)$. Next, we will define the predicate $\mathbf{V}(\cdot)$.

Definition 3.3 (Payload validation predicate). Consider a blockchain C of length $\ell \in \mathbb{N}$, with payload $x_{\mathsf{C}} = \langle x_1, \ldots, x_{\ell} \rangle$. We say deterministic $\mathbf{V}(\cdot)$ is a *payload validation predicate* for blockchain C, if all the following conditions hold (1) $\mathbf{V}(\epsilon) = 1$, (2) if $\mathbf{V}(x_{\mathsf{C}}) = 1$, then there exists a new payload x such that $\mathbf{V}(x_{\mathsf{C}}, x) = 1$.

The first constraint says that blockchain development must be started in a correct initial state. The second constraint means that making progress in constructing a blockchain is always possible. With these constrains applied, a blockchain protocol can achieve its properties (namely Chain Growth, Common Prefix and Chain Quality). In this section, we leave predicate $\mathbf{V}(\cdot)$ undetermined. In following sections we will instantiate the predicate $\mathbf{V}(\cdot)$ with a concrete blockchain application in mind.

3.3 Stable and unstable blockchain payload.

We term the sequence of all payloads except the last κ payloads (the confirmed portion) as the *stable blockchain payload*, and the sequence of the last κ payloads (the unconfirmed portion) as the *unstable blockchain payload.* We formally define these terms as follows.

Definition 3.4 (Stable and unstable blockchain payload). Let $\mathsf{B}_\emptyset, \mathsf{B}_1, \ldots, \mathsf{B}_\ell$ be the ordered sequence of blocks in a given round, for $\ell \in \mathbb{N}$, where $\mathsf{B}_j = \langle \mathsf{head}_j, x_j \rangle$, for $j \in [1, \ldots, \ell]$. Let κ be the security parameter. We then say $\langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$ is a *stable blockchain payload* and $\langle x_{\ell-\kappa+1}, x_{\ell-\kappa+2}, \ldots, x_\ell \rangle$ is an *unstable blockchain payload.*

We recall that a blockchain C is constituted by processing a sequence of blocks, $\mathsf{B}_\emptyset, \mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_\ell$. However, for the sake of presentation, we structure the blockchain in a different but equivalent way. In more detail, we decompose the blockchain C of length ℓ into three components: (1) *the header-chain* (denoted $\mathcal{H}_\ell$), (2) *the stable blockchain payload* (denoted $\bar{x}_\mathsf{C}$), and (3) *the unstable blockchain payload* (denoted $\tilde{x}_\mathsf{C}$); here, the header-chain $\mathcal{H}_\ell := \langle \mathsf{head}_1, \ldots, \mathsf{head}_\ell \rangle$, the stable payload $\bar{x}_\mathsf{C} := \langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$, and the unstable blockchain payload $\tilde{x}_\mathsf{C} := \langle x_{\ell-\kappa+1}, x_{\ell-\kappa+2}, \ldots, x_\ell \rangle$. That is, we can write a blockchain C of length ℓ as $\mathsf{C} := \langle \mathcal{H}_\ell, \bar{x}_\mathsf{C}, \tilde{x}_\mathsf{C} \rangle$. Note that, for the sake of presentation, we use the notation $\mathsf{C}[j, m]$ to indicate $\mathcal{H}_\ell[j, m]$, for $j \geq 1$ and $m \leq \ell$, in the next section. We also abuse $\mathsf{C} \leq \mathsf{C}'$ (where $\mathsf{C}' := \langle \mathcal{H}'_\ell, \bar{x}'_\mathsf{C}, \tilde{x}'_\mathsf{C} \rangle$) to say $\mathcal{H}_\ell \leq \mathcal{H}'_\ell$.

Extending the stable and unstable blockchain payload. We then investigate how to extend the unstable blockchain payload. As defined, the unstable blockchain payload of a blockchain C of length ℓ is formed and updated via a sequence of the last κ payloads, i.e., $\tilde{x}_\mathsf{C} := \langle x_{\ell-\kappa+1}, x_{\ell-\kappa+2}, \ldots, x_\ell \rangle$. In addition, the number of payloads to form the unstable blockchain payload sequence is defined by the security parameter κ. Therefore, whenever a new block $\mathsf{B}_{\ell+1}$ with a new payload $x_{\ell+1}$ is introduced in the system, the oldest payload in the unstable blockchain payload no longer belongs to the last κ payloads. This payload will therefore be appended to the stable blockchain payload. In the meantime, the new coming payload $x_{\ell+1}$ is appended to the unstable blockchain payload.

This intuition is captured by the operation $\blacktriangle$, which is formally defined in Algorithm 1. Specifically, for the sequence $\tilde{x}_\mathsf{C} := \langle x_{\ell-\kappa+1}, x_{\ell-\kappa+2}, \ldots, x_\ell \rangle$, the operation appends the new payload $x_{\ell+1}$, and then removes the payload $x_{\ell-\kappa+1}$ from the sequence to produce an updated sequence $\tilde{x}'_\mathsf{C} := \langle x_{\ell-\kappa+2}, \ldots, x_\ell, x_{\ell+1} \rangle$, and then returns a payload $x_{\ell-\kappa+1}$. Please refer to Algorithm 1.

Algorithm 1 Operation $\blacktriangle$.

```
function ▲ ( $\tilde{x}_\mathsf{C}, x_{\ell+1}$ )
    $\langle x_{\ell-\kappa+1}, x_{\ell-\kappa+2}, \ldots, x_\ell \rangle \leftarrow \tilde{x}_\mathsf{C}$
    $\tilde{x}'_\mathsf{C} := \langle x_{\ell-\kappa+2}, \ldots, x_\ell, x_{\ell+1} \rangle$    ▷ concatenate $x_{\ell+1}$ and remove $x_{\ell-\kappa+1}$
    return $\langle \tilde{x}'_\mathsf{C}, x_{\ell-\kappa+1} \rangle$
end function
```

Compressed stable blockchain payload. Recall that we treat the blockchain C as a set of a header-chain, a stable blockchain payload, and an unstable blockchain payload, i.e., $\mathsf{C} = \langle \mathcal{H}_\ell, \bar{x}_\mathsf{C}, \tilde{x}_\mathsf{C} \rangle$. As the next step, we allow miners to compress the stable blockchain payload of the blockchain C (of length ℓ) to a succinct version, called *compressed stable blockchain payload* and denoted $\bar{\mathbf{x}}_\mathsf{C}$. Now, the blockchain C consists of (1) a header-chain, (2) a compressed stable blockchain payload, and (3) an unstable blockchain payload , i.e., $\mathsf{C} := \langle \mathcal{H}_\ell, \bar{\mathbf{x}}_\mathsf{C}, \tilde{x}_\mathsf{C} \rangle$.

In more detail, the stable blockchain payload $\bar{x}_\mathsf{C} := \langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$ is compressed to $\bar{\mathbf{x}}_\mathsf{C}$ via an operation "$\diamond$". We write, $\bar{\mathbf{x}}_\mathsf{C} := x_1 \diamond x_2 \diamond \ldots \diamond x_{\ell-\kappa}$. We stress that, this operation is instantiated later depending on the application on top of the blockchain. For example, if miners want to keep all payloads of the blockchain, the operation $\diamond$ can be instantiated as a concatenation operation, i.e., "$||$". On the other hand, if miners want to only store *partial* information, it can be instantiated as the operation $\circ$ (see Section 6.1). For the sake of presentation, we defer the details of these operations to next sections. Note that, the compressed stable blockchain payload $\bar{\mathbf{x}}_\mathsf{C}$ is initially defined by $\bar{\mathbf{x}}_\emptyset$; here, $\bar{\mathbf{x}}_\emptyset$ could be set as empty, or it may contain some initial information needed for an application.

Here, the compressed stable blockchain payload is validated via a *compressed payload validation predicate.* Formally, the compressed payload validation predicate is defined as follows.

Definition 3.5 (Compressed payload validation predicate). Consider a chain $\mathsf{C} := \langle \mathcal{H}_\ell, \bar{\mathbf{x}}_\mathsf{C}, \tilde{x}_\mathsf{C} \rangle$ of length $\ell \in \mathbb{N}$, with an operation "$\diamond$". Let $\bar{\mathbf{x}}_\emptyset$ be the initial compressed stable blockchain payload. We say deterministic $\mathsf{V}(\cdot)$ is a *compressed payload validation predicate*, if the following conditions hold (1) $\mathsf{V}(\bar{\mathbf{x}}_\emptyset) = 1$, (2) if $\mathsf{V}(\bar{\mathbf{x}}_\mathsf{C}) = 1$, then there exists a new payload x such that $\mathsf{V}(\bar{\mathbf{x}}_\mathsf{C}, x) = 1$.

3.4 Security properties for the blockchain

In [5] (and then [9, 16], several important security properties, *common prefix property* [5, 16], *chain quality property* [5], and *chain growth property* [9], have been defined for Bitcoin blockchain protocols. Please refer to Appendix A for details on the execution of the blockchain protocol.

Definition 3.6 (Chain growth). Consider a blockchain protocol Π_C among a set $\mathcal{P}$ of players. The chain growth property Q_{cg} with parameter $g \in \mathbb{R}$ states that for any honest player $P \in \mathcal{P}$ with the local chain C of length ℓ in round r and C' of length ℓ' in round r', where $r' - r > 0$, in $\textsc{View}_{\Pi_\mathsf{C}, \mathcal{A}, \mathcal{Z}}$. It holds that $\ell' - \ell \geq g \cdot (r' - r)$ for $g > 0$.

Definition 3.7 (Chain quality). Consider a blockchain protocol Π_C among a set $\mathcal{P}$ of players. The chain quality property Q_{cq} with parameters $\mu \in \mathbb{R}$ and $T \in \mathbb{N}$ states that for any honest player $P \in \mathcal{P}$ with a local chain C of length ℓ in $\textsc{View}_{\Pi_\mathsf{C}, \mathcal{A}, \mathcal{Z}}$, it holds that for large enough T consecutive blocks of C the ratio of honest headers is at least μ.

Definition 3.8 (Common prefix). Consider a blockchain protocol Π_C among a set $\mathcal{P}$ of players. The common prefix property Q_{cp} with parameter $\kappa \in \mathbb{N}$ states that for any two honest players, P' with a best local chain C' of length ℓ' in round r', and P'' with a best local chain C'' of length ℓ'' in round r'', in $\textsc{View}_{\Pi_\mathsf{C}, \mathcal{A}, \mathcal{Z}}$

where $P', P'' \in \mathcal{P}$, $r' \leq r''$, it holds that $\mathrm{C}'[1, l] \preceq \mathrm{C}''$ where $l = \ell' - \kappa$.

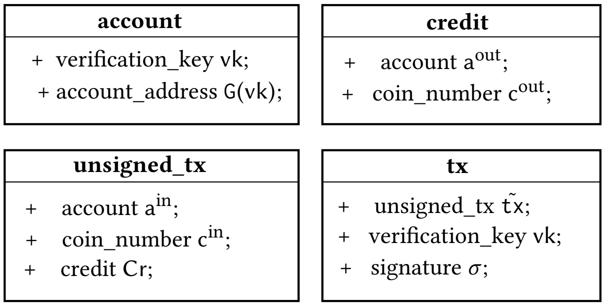
Jumping ahead, the three properties can also be defined for the header-chain.

4 LEDGER AND ITS GENERALIZATION

4.1 Preliminary: Transaction

Transaction generation and verification. Transactions are datagrams that are produced and consumed by the users. By a *digital signature scheme* [7], the users are able to create accounts and sign *transactions*.

Transaction format and data structure. We are now giving more details on our proposed data structures for account, credit, signed and unsigned transactions in Figure 1. The account contains a verification key vk, and its address $\mathsf{G}(\mathsf{vk})$. The credit stores an output account and the number of coins $\mathrm{c}^{\mathrm{out}}$; thus, $\mathrm{c}^{\mathrm{out}}$ will be transfered from an input account to $\mathrm{a}^{\mathrm{out}}$. The unsigned transaction includes a debited/input account, and the number of debited coins c^{in} from the input account. The signed transaction consists of the corresponding unsigned transaction, a verification key and the signature of this transaction.



account

+ verification_key vk;
+ account_address G(vk);

credit

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

unsigned_tx

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

tx

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

Figure 1: Transaction structure

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

Non-conflicting transactions. We define what it means for two transactions to be non-conflicting. We assume that 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 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 4.1 (Non-conflicting Transaction). Consider a chain C of the length $\ell \in \mathbb{N}$ which contains a blockchain payload $x_{\mathrm{C}} := \langle \mathtt{tx}_1, \ldots, \mathtt{tx}_m \rangle$ for some $m \in \mathbb{N}$. We say a transaction $\mathtt{tx}$ is *not conflicting* with x_{C} if and only if predicate $\mathsf{NonConflict}(x_{\mathrm{C}}, \mathtt{tx}) = 1$, where

$$\mathsf{NonConflict}(x_{\mathrm{C}}, \mathtt{tx}) = \begin{cases} 1 & \forall i \in [m],\ \mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathrm{a}^{\mathrm{in}} \neq \mathtt{tx}.\tilde{\mathtt{tx}}.\mathrm{a}^{\mathrm{in}} \\ 0 & \textit{otherwise} \end{cases}$$

Traceable transactions. 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}).

Definition 4.2 (Traceable Transaction). Consider a chain C of the length $\ell \in \mathbb{N}$ which contains a blockchain payload $x_{\mathrm{C}} := \langle \mathtt{tx}_1, \ldots, \mathtt{tx}_m \rangle$ for some $m \in \mathbb{N}$. We say a transaction $\mathtt{tx}$ is *traceable* with respect to x_{C} if and only if predicate $\mathsf{Traceable}(x_{\mathrm{C}}, \mathtt{tx}) = 1$, where

$$\mathsf{Traceable}(x_{\mathrm{C}}, \mathtt{tx}) = \begin{cases} 1 & \exists i \in [m], \exists t \in [\ell_i]\ \textit{s.t.} \\ & (1), \mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathrm{Cr}.\mathrm{a}^{\mathrm{out}}_t = \mathtt{tx}.\tilde{\mathtt{tx}}.\mathrm{a}^{\mathrm{in}}, \\ & \textit{and } (2), \mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathrm{Cr}.\mathrm{c}^{\mathrm{out}}_t = \mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathrm{c}^{\mathrm{in}} \\ 0 & \textit{otherwise} \end{cases}$$

where ℓ_i is the number of output accounts in transaction $\mathtt{tx}_i$.

Spent and unspent transaction output accounts. We next define two types of transaction output accounts that are useful for our construction in next section, namely, *spent transaction output account* and *unspent transaction output account*. Roughly speaking, a transaction output account is considered as spent if it becomes a transaction input account in a later transaction. On the other hand, if it is not a transaction input account of any following transactions, we call it an unspent transaction output account. To simplify the presentation, we simply call transaction input accounts as transaction inputs. Similarly, we call transaction output accounts as transaction outputs. We formally define them as follows.

Definition 4.3 (Spent Transaction Output). Consider a chain C of the length $\ell \in \mathbb{N}$ which contains a blockchain payload $x_{\mathrm{C}} := \langle \mathtt{tx}_1, \ldots, \mathtt{tx}_m \rangle$ for some $m \in \mathbb{N}$. We say a transaction output $\mathrm{a}^{\mathrm{out}}$ of a transaction $\mathtt{tx}_j$ for $j \in [m]$is *spent* with respect to x_{C} if and only if predicate $\mathsf{Spent}(x_{\mathrm{C}}, \mathrm{a}^{\mathrm{out}}) = 1$, where

$$\mathsf{Spent}(x_{\mathrm{C}}, \mathrm{a}^{\mathrm{out}}) = \begin{cases} 1 & \exists i \in [m],\ \mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathrm{a}^{\mathrm{in}} = \mathrm{a}^{\mathrm{out}} \\ 0 & \textit{otherwise} \end{cases}$$

Definition 4.4 (Unspent Transaction Output). Consider a chain C of the length $\ell \in \mathbb{N}$ which contains a blockchain payload $x_{\mathrm{C}} := \langle \mathtt{tx}_1, \ldots, \mathtt{tx}_m \rangle$ for some $m \in \mathbb{N}$. We say a transaction output $\mathrm{a}^{\mathrm{out}}$ of a transaction $\mathtt{tx}_j$ for $j \in [m]$is *unspent* with respect to x_{C} if and only if predicate $\mathsf{UnSpent}(x_{\mathrm{C}}, \mathrm{a}^{\mathrm{out}}) = 1$, where

$$\mathsf{UnSpent}(x_{\mathrm{C}}, \mathrm{a}^{\mathrm{out}}) = \begin{cases} 1 & \forall i \in [m],\ \mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathrm{a}^{\mathrm{in}} \neq \mathrm{a}^{\mathrm{out}} \\ 0 & \textit{otherwise} \end{cases}$$

4.2 Preliminary: Ledger

Since we are interested in transactions as the content of the ledger, the payload x is instantiated as a set of transactions, $x := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$ for some $e \in \mathbb{N}$. We write $\mathtt{tx} \in x$ to denote a transaction $\mathtt{tx}$ that is in the transaction set x. If we particularly consider a transaction set in the i-th block, we write x_i.

The input inserted at each block of the chain C is a set of transactions, i.e., $x := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$. We then term the stable

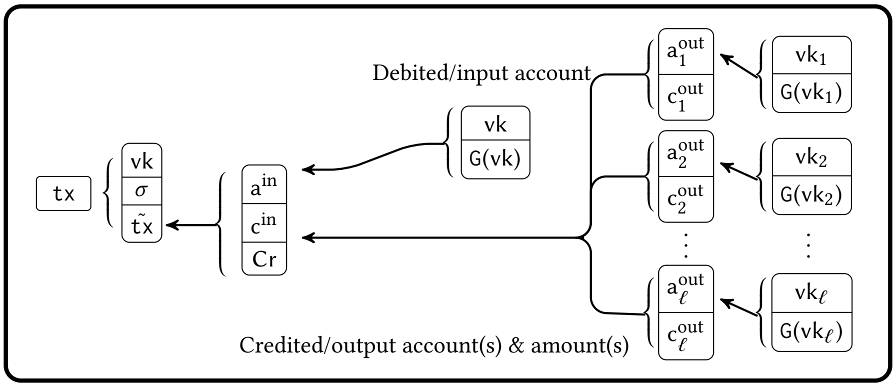


Figure 2: Transaction Data-structure Hierarchy, where $\mathsf{c}^{\mathrm{in}} = \sum_{i=1}^{\ell} \mathsf{c}_i^{\mathrm{out}}$.

blockchain payload (in this case, the stable blockchain transaction) of the chain as *ledger*. Let κ denote the security parameter. We formally present the definition of a ledger as follows.

Definition 4.5 (Ledger). Let κ be the security parameter. Consider a chain C of the length $\ell \in \mathbb{N}$ which contains a blockchain payload $x_{\mathsf{C}} := \langle x_1, \ldots, x_\ell \rangle$. We say $\mathcal{L}_\ell$ is the ledger (of length ℓ) of C, if $\mathcal{L}_\ell = \langle x_1, \ldots, x_{\ell-\kappa} \rangle$.

Informally, we have a ledger of length $\ell \in \mathbb{N}$ consists of all transactions except the last κ transaction sets, i.e., $x_{\mathsf{C}} := \langle x_1, \ldots, x_{\ell-\kappa} \rangle$, where x_i is the input of the i-th block in C and $x_i := \langle \mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_e \rangle$. In addition, it should hold that $\mathbf{V}(\mathcal{L}_\ell) = 1$, where $\mathbf{V}(\cdot)$ is a payload validation predicate (Definition 3.3).

Here we follow the work by Pass et al. [16], and define the stable blockchain payload as the ledger. That is, in a ledger, the last κ number of transaction sets of the blockchain will be simply truncated.

4.3 Generalizing ledger: Snapshot

We here relax the definition of ledger so that more efficient realizations can be allowed. Note that, for many important applications such as cryptocurrency, a properly relaxed version of ledger may be sufficient.

Snapshot. Now, we still consider input inserted at each generalized block of the generalized chain C is a sequence of transactions; that is, $x := \langle \mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_e \rangle$. However, from the generalized chain C of the length ℓ, we obtain a generalized ledger, called *snapshot* and denoted SS_ℓ. Informally, the snapshot is constituted by applying a general operation $\diamond$ to the stable blockchain payload (in this case, stable blockchain transaction— the sequence of transaction sets *truncating the last κ transaction sets*), i.e., $\mathsf{SS}_\ell = x_1 \diamond \ldots \diamond x_{\ell-\kappa}$, where $x_i := \langle \mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_e \rangle$ is the input. In other words, the snapshot SS_ℓ is the compressed stable blockchain payload. Moreover, it should hold that $\mathsf{V}(\mathsf{SS}_\ell) = 1$, where $\mathsf{V}(\cdot)$ is a compressed payload validation predicate (Definition 3.5). We formally define a snapshot as follows.

Definition 4.6 (Snapshot). Let κ be the security parameter. Consider a generalized chain $\mathsf{C} := \langle \mathcal{H}_\ell, \bar{\mathbf{x}}_{\mathsf{C}}, \tilde{x}_{\mathsf{C}} \rangle$ of length $\ell \in \mathbb{N}$, with an operation "$\diamond$" , where $\bar{\mathbf{x}}_{\mathsf{C}} := x_1 \diamond x_2 \diamond \ldots \diamond x_{\ell-\kappa}$. We say SS_ℓ is the snapshot (of length ℓ) of C, if $\mathsf{SS}_\ell = \bar{\mathbf{x}}_{\mathsf{C}}$.

Generalized operation rules. The snapshot SS_ℓ of C is constituted by a sequence of transaction sets via the operation"$\diamond$". We call the operation along with the compressed payload validation predicate $\mathsf{V}(\cdot)$, *an operation rule*, denoted $(\diamond, \mathsf{V}(\cdot))$. More formally, we define *an operation rule* as follows.

Definition 4.7 (Generalized operation rule). Let $\mathsf{B}_\emptyset, \mathsf{B}_1, \ldots, \mathsf{B}_\ell$ be the ordered sequence of blocks. Consider a generalized blockchain C of the length $\ell \in \mathbb{N}$, with the corresponding snapshot SS_ℓ and the operation $\diamond$. Let $\mathsf{V}(\cdot)$ be a compressed payload validation predicate. Let κ be the security parameter. We say $(\diamond, \mathsf{V}(\cdot))$ is a *generalized operation rule* if for any generalized chain C, it holds that (1) $\mathsf{SS}_\ell = x_1 \diamond \ldots \diamond x_{\ell-\kappa}$ where x_i is the payload of block B_i for $i \in [1, \ldots, \ell]$, and (2) $\mathsf{V}(\mathsf{SS}_\ell) = 1$.

Security properties for snapshot. Once the snapshot has been defined we inext nvestigate how to define the relaxed security properties for the ledger protocol when nodes are running in the snapshot mode to snapshot persistence and snapshot liveness.

Snapshot-persistence. Intuitively, the snapshot-persistence property implies, during a time period, the recorded information (i.e., transactions) on two different snapshots of any pair of honest parties should be consistent with each other. This means if a snapshot of length ℓ of a player is progressed by a transaction set x, then snapshots of any other players at *the same length* should also be progressed by *the same transaction set*. Formally, we state the definition as follows.

Definition 4.8 (Snapshot persistence). Consider a generalized ledger protocol $\Pi_{\mathcal{L}}$ with the operation rule $(\diamond, \mathsf{V}(\cdot))$. Let κ be the security parameter. Let Δ denote the upper bound on network latency. Let SS_ℓ^1 be the resulting snapshot at round r_1 of the length ℓ reported by player P_1, for $\ell \in \mathbb{N}$ and $\ell > \kappa$.

Here snapshot persistence property states that, if snapshot SS_ℓ^1 is extended by a transaction set x such that $\mathsf{SS}_{\ell+1}^1 := \mathsf{SS}_\ell^1 \diamond x$, then for any player P_2 with the reported snapshot SS_ℓ^2 (by player P_2) of the length ℓ, in a round r_2 where $r_2 \geq r_1 + \Delta$, it holds that $\mathsf{SS}_{\ell+1}^2 := \mathsf{SS}_\ell^2 \diamond x$.

Snapshot-liveness. Intuitively, the liveness guarantees that any new transactions, that do not conflict with the recorded information, will be definitely recorded on the ledger after a certain number of rounds. Similarly, snapshot liveness guarantees that the new transactions, if they do not conflict with the snapshot, will be accepted and incorporated into the updated snapshot, after a certain number of rounds.

Definition 4.9 (Snapshot liveness). Consider a generalized ledger protocol $\Pi_{\mathcal{L}}$ with the operation rule $(\diamond, \mathsf{V}(\cdot))$. Let κ be the security parameter. Consider a "wait time" parameter t.

Here snapshot liveness property states that, if a valid transaction $\mathtt{tx}$ is given as input to every honest player P continuously for t rounds from a given round r, then there exist snapshots $\mathsf{SS}_{\ell}, \mathsf{SS}_{\ell+1}$ and a transaction set x (reported by P) of the length $\ell > \kappa$, in round $r' \leq r + t$, such that $\mathsf{SS}_{\ell+1} := \mathsf{SS}_{\ell} \diamond x$ and $\mathtt{tx} \in x$.

4.4 From single mode to multi-mode

In a multi-mode protocol, denoted Π^{multi}, there exist multiple types of modes. These modes should be compatible with each other so that the system can run stably and securely. Moreover, each mode is associated with an operation rule. Here, each operation rule is defined by an operation and *a multi-mode snapshot validation predicate*. Consider a mode i where $i \in [m]$, the operation rule in this mode is defined as $(\diamond_i, \mathbb{V}_i)$, where $\mathbb{V}_i$ denotes a multi-mode snapshot validation predicate and $\diamond_i$ denotes the general operation in mode i.

Similar to the compressed payload validation predicate in Section 3, we require constraints on *the multi-mode snapshot validation predicate*. In addition to the constrains for the compressed payload validation predicate, a multi-mode snapshot validation predicate in the multi-mode system should hold that, different snapshots generated from the same stable blockchain payload (transaction) should be rejected/accepted in the same way, and also always able to make progress. We formally define a multi-mode snapshot validation predicate as follows.

Definition 4.10 (Multi-mode snapshot validation predicate). Consider a multi-mode ledger protocol Π^{multi} with m modes $\mathsf{M}_1, \mathsf{M}_2, \ldots, \mathsf{M}_m$. For each mode M_i with operation rule $(\diamond_i, \mathbb{V}_i(\cdot))$, the corresponding initial snapshot is $\mathsf{SS}_{\emptyset}^{\mathsf{M}_i}$. Consider a generalized blockchain C of length ℓ, with the corresponding snapshot $\mathsf{SS}_{\ell}^{\mathsf{M}_i}$, in mode i-th, where $i \in [m]$. We say $(\mathbb{V}_1, \ldots, \mathbb{V}_m)$ are a set of *multi-mode snapshot validation predicates*, if the following conditions hold for the same generalized blockchain C

- for any mode M_i, where $i \in [m]$, it holds that $\mathbb{V}_i(\mathsf{SS}_{\emptyset}^{\mathsf{M}_i}) = 1$.
- for any mode M_i, where $i \in [m]$, it holds that, if $\mathbb{V}_i(\mathsf{SS}_{\ell}^{\mathsf{M}_i}) = 1$, then there exists a new payload x such that $\mathbb{V}_i(\mathsf{SS}_{\ell}^{\mathsf{M}_i}, x) = 1$,
- for any pair of modes M_i and M_j, where $i, j \in [m]$, it holds that $\mathbb{V}_i(\mathsf{SS}_{\ell}^{\mathsf{M}_i}) = \mathbb{V}_j(\mathsf{SS}_{\ell}^{\mathsf{M}_j})$.

A multi-mode system is composed by m modes where each mode-i, for $1 \leq i \leq m$, is an instantiation of the generalized ledger with the operation rule $(\diamond_i, \mathbb{V}_i(\cdot))$; concretely, for each mode, we need to instantiate the content of the state of snapshot, the content of the snapshot, along with its operation rule $(\diamond_i, \mathbb{V}_i(\cdot))$.

Defining security for incorporating multiple modes. There are multiple types of nodes, running in different modes, in our multi-mode protocol. Intuitively, these nodes should be compatible. We formalize this intuition via a security property, *multi-mode system soundness*. More concretely, we need to provide the security guarantee that new transactions should be accepted/rejected in the same way by all nodes in different modes. That means, when applying the operation rule to the i-th mode nodes and to the j-th mode nodes, where $i, j \in [m]$, the output should be the same. This property can be trivially achieved if there is only a single mode. However, this property is critical for designing multi-mode cryptocurrencies that has more than one mode (e.g., PRUNE system in the next section). Without this security requirement, trivial protocols could be allowed.

Definition 4.11 (Multi-mode soundness). Consider a multi-mode ledger protocol Π^{multi}. Let m be the total number of modes in Π^{multi}. Consider any pair of a mode i-th with the multi-mode operation rule $(\diamond_i, \mathbb{V}_i(\cdot))$ and a mode j-th with the multi-mode operation rule $(\diamond_j, \mathbb{V}_j(\cdot))$ in Π^{multi}, where $j, i \in [m]$. Let κ be the security parameter. Let Δ denote the upper bound on network latency. Let $\mathsf{SS}_{\ell}^{\mathsf{M}_i}$ be the snapshot of player P_1 in the mode i-th of the length $\ell > \kappa$ (note that, $\ell \in \mathbb{N}$) at round r_1.

Here multi-mode soundness property states that, for any player P_2 in the mode j-th with resulting snapshot $\mathsf{SS}_{\ell}^{\mathsf{M}_j}$ at round $r_2 \geq r_1 + \Delta$ of the length ℓ, it holds that, $\mathbb{V}_i(\mathsf{SS}_{\ell}^{\mathsf{M}_i}, x) = \mathbb{V}_j(\mathsf{SS}_{\ell}^{\mathsf{M}_j}, x)$, for any input x.

5 FULL-MODE

5.1 Instantiating snapshot to FULL-mode

We use Π^{FULL} to denote the ledger protocol in the FULL-mode. In Π^{FULL}, players are expected to store full sequence of historical transaction sets. Please refer to Table 1 for how we instantiate to the FULL-mode, and please also refer to Figure 3 for an illustration of Table 1.

Table 1: Snapshot Instantiation: FULL-mode Π^{FULL} over the blockchain protocol Π_{C} (see Appendix A.1) via the specification of SS_{ℓ}, $\diamond$, and $\mathbb{V}(\cdot)$.

SS_{ℓ}	$\mathcal{T}_{\ell}$ where $\mathcal{T}_{\ell}$ is a full sequence of transactions (truncating the last κ transaction sets), i.e., $\mathcal{T}_{\ell} := \langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$.
$\diamond$	$\|\|$
$\mathbb{V}(\cdot)$	$\mathbb{V}_{\text{FULL}}$ operates in the following way: return true if the argument is ϵ. If the input is $(\mathcal{T}_{\ell})$: – Parse $\mathcal{T}_{\ell} = \langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$, which is a sequence of transactions where $\mathcal{T}_{\ell} := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_m \rangle$ for some $m \in \mathbb{N}$. – $\mathbb{V}_{\text{FULL}}$ outputs true if and only if for all $i \in [m]$, it holds that • $\mathsf{Verify}(\mathtt{tx}_i) = 1$, • $\mathsf{NonConflict}(x_{\mathsf{C}}^{i}, \mathtt{tx}_i) = 1$, • and $\mathsf{Traceable}(x_{\mathsf{C}}^{i}, \mathtt{tx}_i) = 1$, where $x_{\mathsf{C}}^{i} := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_{i-1} \rangle$ and $x_{\mathsf{C}}^{0} = \emptyset$ initially. If the input is $(\mathcal{T}_{\ell}, x)$: $\mathbb{V}_{\text{FULL}}$ outputs true if and only if, – $\mathbb{V}_{\text{FULL}}(\mathcal{T}_{\ell}) = 1$, – and $\mathbb{V}_{\text{FULL}}(\mathcal{T}_{\ell+1}) = 1$, where $\mathcal{T}_{\ell+1} := \mathcal{T}_{\ell} \|\| x$.

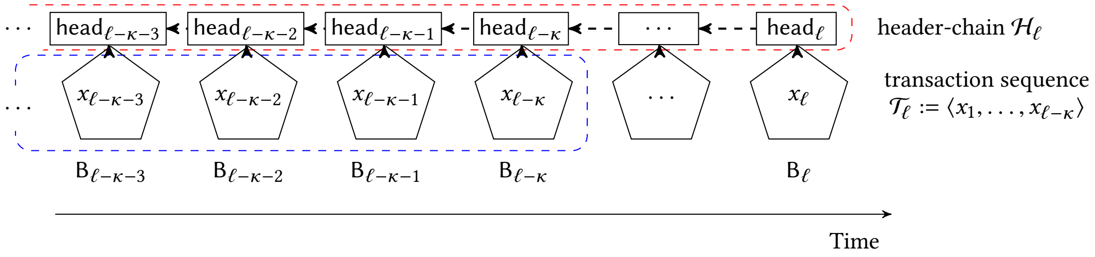

$\cdots$
$\mathsf{head}_{\ell-\kappa-3}$
$\mathsf{head}_{\ell-\kappa-2}$
$\mathsf{head}_{\ell-\kappa-1}$
$\mathsf{head}_{\ell-\kappa}$
$\cdots$
head_{ℓ}
header-chain $\mathcal{H}_\ell$
$\cdots$
$x_{\ell-\kappa-3}$
$x_{\ell-\kappa-2}$
$x_{\ell-\kappa-1}$
$x_{\ell-\kappa}$
$\cdots$
x_ℓ
transaction sequence
$\mathcal{T}_\ell := \langle x_1, \ldots, x_{\ell-\kappa} \rangle$
$\mathsf{B}_{\ell-\kappa-3}$
$\mathsf{B}_{\ell-\kappa-2}$
$\mathsf{B}_{\ell-\kappa-1}$
$\mathsf{B}_{\ell-\kappa}$
B_{ℓ}
Time

Figure 3: The illustration of the blockchain in FULL-mode.

Since we are interested in transactions as the content of the ledger, we instantiate the payload x in a block as a set of transactions, and write $x := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$. We use x_i to denote the payload of the i-th block. We then have a full sequence of transaction sets of the length ℓ except the last κ transaction sets $\mathcal{T}_\ell := \langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$. Intuitively, if there is a new transaction set (returned by operation ▲, see Algorithm 1), then the sequence of transaction sets is concatenated by one more transaction set such that the new transaction sequence is valid with respect to a multi-mode snapshot validation predicate. Note that, we need instantiate the predicate $\mathbb{V}(\cdot)$ to $\mathbb{V}_{\text{FULL}}(\cdot)$ in this FULL-mode. In more detail, from Table 1, for any transaction set x (returned by operation ▲), it should follow that $\mathsf{SS}_{\ell+1} := \mathsf{SS}_\ell \diamond x$ and $\mathbb{V}_{\text{FULL}}(\mathsf{SS}_{\ell+1})$, where $\diamond := ||$. This implies, $\mathcal{T}_{\ell+1} := \mathcal{T}_\ell || x = \langle x_1, x_2, \ldots, x_{\ell-\kappa}, x \rangle$; here, the updated transaction sequence is valid with respect to data validation predicate $\mathbb{V}_{\text{FULL}}(\cdot)$, i.e., $\mathbb{V}_{\text{FULL}}(\mathcal{T}_{\ell+1}) = 1$ (see Table 1). In a nutshell, the predicate $\mathbb{V}_{\text{FULL}}(\cdot)$ checks (1) the conflict between any pair of transactions, and (2) the integrity of every transaction from its input. Details are provided in Table 1.

5.2 Security analysis for FULL-mode

We begin by showing that the execution in FULL-mode satisfies the snapshot persistence, with an overwhelming probability in κ (see Section 4.4, Definition 4.8). The proof is essentially based on the common-prefix property of the underlying header-chain and the collision-resistance of the authenticated data structure used in this mode.

Technical challenges. Different from the proof in [5], we need the collision-resistance of the authenticated data structure to show the security since our blockchain data structure is different from theirs. More precisely, we include the one-way digest τ of transaction set x into the header-chain. Therefore, we must ensure that each transaction set corresponds to only a single header. In other words, there should not be two different transaction sets having the same header. If there exist those transaction sets, adversarial miners can always rewrite the confirmed and processed transactions. This is guaranteed by the collision-resistance of the authenticated data structure. Thus, we need to reduce the security of the snapshot protocol to the collision-resistance of the authenticated data structure. We first formally state the theorems as follows.

Theorem 5.1 (FULL-mode persistence). *Consider the protocol Π^{FULL} (see Section 5.1). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition 3.1) is collision-resistant. Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. Then, it holds that the execution in FULL-mode Π^{FULL} satisfies the snapshot persistence property, with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.*

Proof can be found in Appendix B.1.

Note that snapshot persistence property (see Section 4.4, Definition 4.9) in FULL-mode is useful but not enough to ensure that the players in FULL-mode makes progress, i.e., that transactions are eventually accepted (or recorded). This is captured by the snapshot liveness property in FULL-mode. The snapshot liveness property in FULL-mode is formally stated in Theorem 5.2. Proof can be found in Appendix B.2.

Theorem 5.2 (FULL-mode liveness). *Consider the protocol Π^{FULL} (see Section 5.1). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition 3.1) is collision-resistant. Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. Then, it holds that the execution in FULL-mode Π^{FULL} satisfies the snapshot liveness property where $t = (1 + \delta)\frac{2\kappa}{\gamma}$, for $\delta > 0$, with probability at least $1 - \epsilon(\kappa)$ where $\epsilon(\cdot)$ is a negligible function.*

6 PRUNE-MODE

6.1 Building block: UTXO-set

Algorithm 2 UTXO operation $\circ$.

```
function ∘ ( $\mathcal{U}_\ell$, $x_{\ell+1}$ )
    $\mathcal{U}_{\ell+1} := \mathcal{U}_\ell$
    for each transaction tx in $x_{\ell+1}$ do        ▷ transactions are stored in temporal order
        for each $a^{in}$ in tx do
            if $a^{in} \notin \mathcal{U}_{\ell+1}$ then
                return ⊥
            else
                remove $a^{in}$ from $\mathcal{U}_{\ell+1}$
            end if
        end for
        add all transaction outputs $a^{out}$ in tx to $\mathcal{U}_{\ell+1}$
    end for
    return $\mathcal{U}_{\ell+1}$
end function
```

In Bitcoin, each output of a particular transaction can only be spent once; the outputs of all transactions included in the blockchain can be categorized as either unspent transaction outputs [2, 12] or spent transaction outputs (see Definition 4.4). For a payment to be

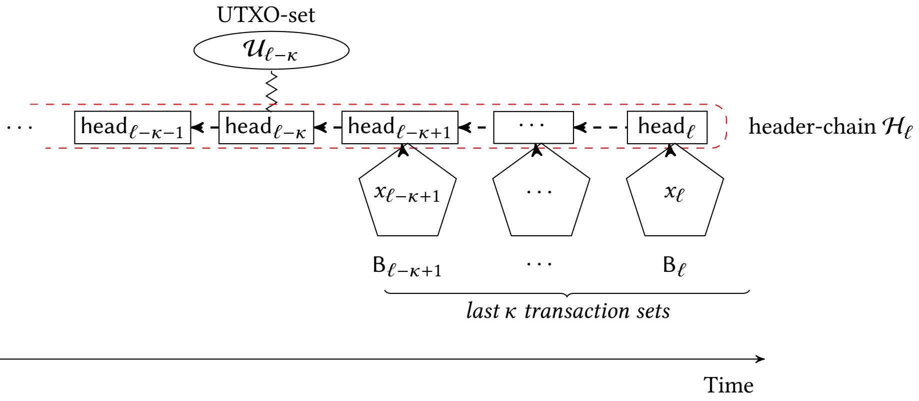

UTXO-set
$\mathcal{U}_{\ell-\kappa}$
· · ·
$\mathsf{head}_{\ell-\kappa-1}$
$\mathsf{head}_{\ell-\kappa}$
$\mathsf{head}_{\ell-\kappa+1}$
· · ·
head_{ℓ}
header-chain $\mathcal{H}_\ell$
$x_{\ell-\kappa+1}$
· · ·
x_ℓ
$\mathsf{B}_{\ell-\kappa+1}$
· · ·
B_ℓ
last κ transaction sets
Time

Figure 4: The illustration of the blockchain in PRUNE-mode.

valid, it must use previously unspent outputs as inputs. Therefore, spent transaction outputs are not necessary to be stored, and a set of unspent transaction outputs could be abstracted and represented as a *UTXO-set*, denoted $\mathcal{U} := \langle \mathsf{a}_1^{\text{out}}, \mathsf{a}_2^{\text{out}}, \ldots, \mathsf{a}_n^{\text{out}} \rangle$ for some $n \in \mathbb{N}$. Now, instead of storing all historical and unnecessary transactions, miners are expected to store a UTXO-set.

Jumping ahead, each UTXO-set corresponds to a blockchain length. Let ℓ denotes the current length. Intuitively, a UTXO-set $\mathcal{U}_i$ of any player P denotes the UTXO set with the length i, where $i \in [\ell]$. An initial UTXO-set, denoted as $\mathcal{U}_0$ can be updated into a new set $\mathcal{U}_1$ by incorporating a set of transactions x_1, i.e., $\mathcal{U}_1 := \mathcal{U}_0 \circ x_1$, where operation $\circ$ is defined by Algorithm 2. Similarly, the first UTXO-set $\mathcal{U}_1$ can be further updated into the second UTXO-set $\mathcal{U}_2$ by incorporating another set of transactions x_2, i.e., $\mathcal{U}_2 := \mathcal{U}_1 \circ x_2$, and further $\mathcal{U}_{\ell+1} := \mathcal{U}_\ell \circ x_{\ell+1}$ for any transaction set $x_{\ell+1}$. If we do not interest in a particular length, we ignore the subscript and write $\mathcal{U}$. Also, if we interest in a particular player $P_j \in \mathcal{P}$, where $j \in [n]$ and n is the total number of miners, we write $\mathcal{U}^j$.

6.2 Instantiating snapshot to PRUNE-mode

Table 2: Snapshot instantiation: PRUNE-mode Π^{PRUNE} over the blockchain protocol Π_{C} (see Appendix A.1) via the specification of SS_ℓ, $\diamond$, and $\mathbb{V}(\cdot)$.

SS_ℓ	$\mathcal{U}_{\ell-\kappa}$ where $\mathcal{U}_{\ell-\kappa}$ is the (ℓ-κ)-th UTXO-set.
$\diamond$	$\circ$
$\mathbb{V}$	$\mathbb{V}_{\text{PRUNE}}()$ operates in the following way: return true if the argument is ϵ. If the input is ($\mathcal{U}_{\ell-\kappa}$): If parse $\mathcal{U}_{\ell-\kappa}$ as a sequence of transaction outputs where $\mathcal{U}_{\ell-\kappa} := \langle \mathsf{a}_1^{\text{out}}, \mathsf{a}_2^{\text{out}}, \ldots, \mathsf{a}_n^{\text{out}} \rangle$ for some $n \in \mathbb{N}$, then returns true. If the input is ($\mathcal{U}_{\ell-\kappa}, x$): Parse x as a sequence of transaction $\langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$ for some $e \in \mathbb{N}$. $\mathbb{V}_{\text{PRUNE}}$ outputs true if and only if for all $i \in [e]$, it holds that $\mathtt{tx}_i.\tilde{\mathtt{tx}}.\mathsf{a}^{\text{in}} \in \mathcal{U}_{\ell-\kappa}$.

This design goal is captured by the following instantiation of the snapshot to PRUNE-mode Π^{PRUNE} (see Table 2). Let $\ell \in \mathbb{N}$ denote the current blockchain length. In Π^{PRUNE}, players are expected to store the $\{\ell$-$\kappa\}$-th UTXO-set. In Table 2, we instantiate the snapshot SS_ℓ of any player P. (We write SS_ℓ^j, if we are interested in the snapshot of a particular player P_j, where $1 \leq j \leq n$.) Please also refer to Figure 4 for an illustration of Table 2.

Let $\mathcal{U}_0$ denote the initial UTXO-set such that $\mathbb{V}_{\text{PRUNE}}(\mathcal{U}_0) = 1$. For $\ell < \kappa$, we set $\mathcal{U}_{\ell-\kappa} := \mathcal{U}_0$. From Table 2, for any new transaction set x (returned by operation $\blacktriangle$, see Algorithm 1), it should follow that $\mathsf{SS}_{\ell+1} := \mathsf{SS}_\ell \diamond x$ and $\mathbb{V}_{\text{FULL}}(\mathsf{SS}_{\ell+1})$, where $\diamond := \circ$. This implies, $\mathcal{U}_{\ell-\kappa+1} := \mathcal{U}_{\ell-\kappa} \circ x$.

Without lost of generality, we assume that the initial UTXO set is valid with respect to the initialization of the ledger in the FULL-mode. Our instantiation of SS_ℓ in the PRUNE-mode follows certain rules which could be specified by a multi-mode snapshot validation predicate $\mathbb{V}_{\text{PRUNE}}$. This predicate is used to validate a snapshot. If it receives as input $\mathcal{U}_{\ell-\kappa}$, the predicate returns `True` if and only if $\mathcal{U}_{\ell-\kappa}$ is a valid UTXO-set to the application's point of view.

6.3 Security analysis for PRUNE-mode

Similar to the proofs for the snapshot persistence and snapshot liveness in FULL-mode Π^{FULL}, the proofs for the snapshot persistence and snapshot liveness properties in PRUNE-mode Π^{PRUNE} (see Section 4.4, Definitions 4.8 and 4.9), are essentially based on the common-prefix property, chain-quality property, and chain growth property for the header-chains together with the collision-resistance of the authenticated data structure used in this mode.

We begin by introducing an important lemma to prove the security. Informally, the lemma states that if any pair of players has the same UTXO-sets, then when the UTXO-sets are extended, they are identical. The lemma is formally state as follows. (Proof can be found in Appendix C.1.)

Lemma 6.1. *Consider the protocol Π^{PRUNE} (see Section 6.2). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition 3.1) is collision-resistant. Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. Assume that $\mathcal{U}^1_{\ell-\kappa} = \mathcal{U}^2_{\ell-\kappa}$, for any two honest players P_1*

and P_2. It holds that $\mathcal{U}^1_{\ell-\kappa+1} = \mathcal{U}^2_{\ell-\kappa+1}$ with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function in κ.

We are now ready to prove the snapshot persistence considering the execution in PRUNE-mode as follows.

Theorem 6.2 (PRUNE-mode persistence). *Consider the protocol Π^{PRUNE} (see Section 6.2). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition 3.1) is collision-resistant. Assume that $\gamma = \lambda\beta$ and $\lambda > 1$, it holds that the execution in PRUNE-mode Π^{PRUNE} satisfies the snapshot persistence property, with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.*

Proof can be found in Appendix C.2.

Theorem 6.3 (PRUNE-mode liveness). *Consider the generalized ledger protocol Π^{PRUNE} (see Section 6.2). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition 3.1) is collision-resistant. Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. It holds that the execution in PRUNE-mode Π^{PRUNE} satisfies the snapshot liveness property with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.*

Proof can be found in Appendix C.3.

7 PRUNE: A MULTI-MODE SYSTEM WITH FULL AND PRUNE MODES

There are two modes, FULL-mode Π^{FULL} (see Section 5), and PRUNE-mode Π^{PRUNE} (see Section 6) in the PRUNE system, and we write $\Pi^{\mathrm{multi}} = (\Pi^{\mathrm{FULL}}, \Pi^{\mathrm{PRUNE}})$. Both modes are instantiations of the generalized ledger. We remark that our PRUNE system can be viewed as an abstract presentation of the Bitcoin Pruned proposal (introduced in Bitcoin Core version 0.11 [1]). We also remark that, our PRUNE can be easily modified to capture the Ethereum Pruned proposal [21]. As an initial study and to simplify the analysis, we assume that players will not change modes in the execution.

7.1 Security analysis for incorporating FULL-mode and PRUNE-mode

In this section, we prove the multi-mode system soundness (see Section 4.4, Definition 4.11). Intuitively, the multi-mode system soundness property says that any transaction should be accepted/rejected in the same way as by the PRUNE-mode players or by FULL-mode players. The proof is essentially based on the snapshot persistence property for PRUNE-mode and FULL-mode. Before going to the details of the proof of multi-mode system soundness, we state a lemma to show that unspent transaction outputs are as useful as the full set of transactions. More precisely, new transactions should be accepted/rejected in the same way with respect to the snapshot in FULL-mode (generated from the full sequence of all processed transactions truncating the last κ transaction sets), and to the snapshot in PRUNE-mode (containing the set of unspent transaction outputs). This implies the multi-mode snapshot validation predicates in both PRUNE and FULL mode should return the same output.

Note that, to distinguish the snapshots and the corresponding operations in FULL-mode from PRUNE-mode, we denote $\mathsf{SS}^{\mathsf{f}}_{\ell}$ and $\diamond^{\mathsf{f}}$ as the snapshot of length ℓ and operation in FULL-mode, respectively. Similarly, we denote $\mathsf{SS}^{\mathsf{p}}_{\ell}$ and $\diamond^{\mathsf{p}}$ as the snapshot and operation in PRUNE mode, respectively. If we are interested in the snapshot of a particular player P_i, we write $\mathsf{SS}^{\mathsf{f},i}_{\ell}$ (or $\mathsf{SS}^{\mathsf{p},i}_{\ell}$).

Lemma 7.1. *Consider PRUNE system $\Pi^{\mathrm{multi}} = (\Pi^{PRUNE}, \Pi^{FULL})$. Let κ be the security parameter. Consider a snapshot $\mathsf{SS}^{\mathsf{f}}_{\ell} := \mathcal{T}_{\ell}$ in FULL mode where $\mathcal{T}_{\ell} := \langle x_1, x_2, \ldots, x_{\ell-\kappa}\rangle$, and $\mathsf{SS}^{\mathsf{p}}_{\ell} = \mathcal{U}_{\ell-\kappa}$ in PRUNE mode, where $\mathcal{U}_{\ell-\kappa} := \mathcal{U}_0 \circ x_1 \circ \ldots x_{\ell-\kappa}$. It holds that for any transaction set x, $\mathbb{V}_{FULL}(\mathsf{SS}^{\mathsf{f}}_{\ell} \diamond^{\mathsf{f}} x) = \mathbb{V}_{PRUNE}(\mathsf{SS}^{\mathsf{p}}_{\ell} \diamond^{\mathsf{p}} x)$ where $\diamond^{\mathsf{f}} := ||$ and $\diamond^{\mathsf{p}} := \circ$.*

Proof can be found in Appendix D.1.

Armed with Lemma 7.1 and Lemma 6.1 in Section 6.3, we are now ready to prove our main theorem as follows.

Theorem 7.2 (Multi-mode soundness). *Consider PRUNE system $\Pi^{\mathrm{multi}} = (\Pi^{PRUNE}, \Pi^{FULL})$ in Section 7. Let κ be the security parameter. Let κ be the security parameter. Assume the authenticated data structure π_{au} (see Definition 3.1) is collision-resistant. Assume that $\gamma = \lambda\beta$ and $\lambda > 1$, it holds that PRUNE system Π^{multi} satisfies the multi-mode soundness property with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.*

Proof can be found in Appendix D.2

REFERENCES

[1] Bitcoin core version 0.11.0 released. https://bitcoin.org/en/release/v0.11.0.

[2] Bitcoin developer guide – UTXO definition. https://bitcoin.org/en/developer-guide#term-utxo.

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

[4] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, March 2014.

[5] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015.

[6] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*, 2017. https://eprint.iacr.org/2016/1048.

[7] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[8] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation (EC)*, pages 365–382, 2016.

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

[10] Aggelos Kiayias and Giorgos Panagiotakos. On trees, chains and fast transactions in the blockchain. Cryptology ePrint Archive, Report 2016/545, 2016. http://eprint.iacr.org/2016/545.

[11] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990.

[12] Andrew Miller. Storing utxos in a balanced merkle tree. 2012. https://bitcointalk.org/index.php?topic=101734.0.

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

[14] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine 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.

[15] Parity Wiki. Warp sync. 2017. https://github.com/paritytech/parity/wiki/Warp-Sync.

[16] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, 2017. https://eprint.iacr.org/2016/454.

[17] Ayelet Sapirstein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. In *International Conference on Financial Cryptography and Data Security*, 2016.

[18] Okke Schrijvers, Joseph Bonneau, Dan Boneh, and Tim Roughgarden. Incentive compatibility of bitcoin mining pool reward functions. In *International Conference on Financial Cryptography and Data Security*, 2016.

[19] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 507–527. Springer, Heidelberg, January 2015.

[20] Parity Technologies. Fast, light, robust ethereum implementation. https://github.com/paritytech/parity.

[21] Parity Wiki. Warp sync snapshot format. https://github.com/paritytech/parity/wiki/Warp-Sync-Snapshot-Format.

[22] Gavin Wood. Ethereum: A secure decentralized transaction ledger. 2014. http://gavwood.com/paper.pdf.

A SUPPORTING MATERIAL FOR SECTION 3

A.1 The execution of miners

The blockchain protocol Π_C. We are now ready to describe the blockchain protocol Π_C (Algorithm 3). First, each miner copies all the new blocks received from the network into his local state. Then the miner updates set of stored blokchains $\mathbb{C}$ with the blocks and selects the best blockchain from the set by calling the *BestChain* function (Algorithm 6). Every updated blockchain must be valid, thus *BestChain* is calling chain validation function *Validate* (Algorithm 4) under the hood. Then the miner tries to extend the best blockchain by running randomized algorithm *Pow* (Algorithm 5) which is issuing a single query to idealized hash function per round. The pseudocode of main miner loop is provided in Algorithm 3.

Algorithm 3 The blockchain protocol Π_C.

```
$\mathbb{C} := \epsilon$
while True do
    $\mathbb{B} \leftarrow$ all blocks from the network.
    $\langle \mathbb{C}, \mathrm{C} \rangle \leftarrow BestChain(\mathbb{C}, \mathbb{B})$
    $\langle x, \tau \rangle \leftarrow$ a new payload and its corresponding digest from the environment.
    $\langle \mathcal{H}_\ell, \bar{\mathbf{x}}_C, \tilde{x}_C \rangle \leftarrow \mathrm{C}$        $\triangleright$ the length of C is $\ell$
    $\langle \mathrm{head}_1, \mathrm{head}_2, \dots, \mathrm{head}_\ell \rangle \leftarrow \mathcal{H}_\ell$
    $\mathrm{head}_{\ell+1} \leftarrow Pow(\mathrm{head}_\ell, \tau)$
    if $\mathrm{head}_{\ell+1} \neq \epsilon$ then
        $\mathrm{B} := \langle \mathrm{head}_{\ell+1}, x \rangle$
        $\mathcal{H}_{\ell+1} := \langle \mathrm{head}_1, \mathrm{head}_2, \dots, \mathrm{head}_\ell, \mathrm{head}_{\ell+1} \rangle$
        $\langle \tilde{x}'_C, x' \rangle := \tilde{x}_C \blacktriangle x$
        $\bar{\mathbf{x}}'_C := \bar{\mathbf{x}}_C \diamond x'$
        $\mathrm{C}' := \langle \mathcal{H}_{\ell+1}, \bar{\mathbf{x}}'_C, \tilde{x}'_C \rangle$
        $\mathbb{C} := (\mathbb{C} \setminus \{\mathrm{C}\}) \cup \{\mathrm{C}'\}$
        BROADCAST(B)
    end if
    round := round + 1
end while
```

Algorithm 5 The proof of work function *Pow*, parametrized by T and hash function $hash(\cdot)$.

```
function $Pow(\mathrm{head}_\ell, \tau)$
    $h \leftarrow hash(\mathrm{head}_\ell)$
    $w \leftarrow \{0,1\}^\kappa$
    $\mathrm{head}_{\ell+1} := \varepsilon$
    if $hash(h, \tau, w) < \mathrm{T}$ then
        $\mathrm{head}_{\ell+1} := \langle h, \tau, w \rangle$
    end if
    return $\mathrm{head}_{\ell+1}$
end function
```

Algorithm 4 Validation function *Validate*, parameterized by a target T, a hash function $hash(\cdot)$, and compressed payload validation predicate $\mathsf{V}(\cdot)$.

```
function $Validate(\ \mathrm{C}\ )$
    $\langle \mathcal{H}_\ell, \bar{\mathbf{x}}_C, \tilde{x}_C \rangle \leftarrow \mathrm{C}$
    $b := \mathsf{V}(\bar{\mathbf{x}}_C)$
    if $b$ then
        $\langle \mathrm{head}_1, \mathrm{head}_2, \dots, \mathrm{head}_\ell \rangle \leftarrow \mathcal{H}_\ell$ $\triangleright$ the length of C is $\ell$
        $h' := 0$
        for each $i$ in $1, \dots, \ell$ do
            parse $\mathrm{head}_i$ into $\langle h_i, \tau_i, w_i \rangle$
            if $(h_i = h') \wedge (hash(\mathrm{head}_i) < \mathrm{T})$ then
                $h' \leftarrow hash(\mathrm{head}_i)$
            else
                return $(b = 0)$
            end if
        end for
    end if
    return $(b)$
end function
```

Algorithm 6 Best chain function *BestChain*.

```
function $BestChain(\mathbb{C}, \mathbb{B})$
    for each $\mathrm{B} \in \mathbb{B}$ (reordered by using hash-links) do
        $\langle \mathrm{head}, x \rangle \leftarrow \mathrm{B}$ $\triangleright$ iterating over $\mathbb{B}$ to append new blocks
        for each C in $\mathbb{C}$ do
            $\langle \mathcal{H}_\ell, \bar{\mathbf{x}}_C, \tilde{x}_C \rangle \leftarrow \mathrm{C}$        $\triangleright$ the length of C is $\ell$
            $\mathcal{H}_{\ell+1} := \langle \mathcal{H}_\ell, \mathrm{head} \rangle$
            $\langle \tilde{x}'_C, x' \rangle := \tilde{x}_C \blacktriangle x$
            $\bar{\mathbf{x}}'_C := \bar{\mathbf{x}}_C \diamond x'$
            $\mathrm{C}' := \langle \mathcal{H}_{\ell+1}, \bar{\mathbf{x}}'_C, \tilde{x}'_C \rangle$
            if $\mathrm{C}' \neq \perp \wedge Validate(\mathrm{C}')$ then
                $\mathbb{C} := (\mathbb{C} \setminus \{\mathrm{C}\}) \cup \{\mathrm{C}'\}$
            end if
        end for
    end for
    return $\langle \mathbb{C}$, C with a longest header-chain in $\mathbb{C} \rangle$
end function
```

A.2 Security analysis for the blockchain

The underlying header-chain in the blockchain is the same as the Bitcoin blockchain in [5] except that we specify each payload x_i as the digest τ_i of a transaction x_i. Indeed, the security of our header-chain is implied from the security of the blockchain in Bitcoin backbone; therefore, by the security of Bitcoin backbone shown in [5], the blockchain protocol Π_C (Section A.1) achieves the three security properties.We begin by recalling the following two quantities introduced in [5, 16]. Consider the total number of players is n, the portion of malicious computing power is ρ, and $p = \frac{\mathrm{T}}{2^\kappa}$ is the probability of success in a single PoW function invocation.

- Let $\alpha = 1 - (1-p)^{(1-\rho)n}$ be the probability that at least one honest players mines a block successfully in a round.

- Let $\beta = \rho np$ be the expected number of blocks that malicious players can find in a round.

Here, when $pn \ll 1$, we have $\alpha \approx (1-\rho)np$, and thus $\frac{\alpha}{\beta} \approx \frac{1-\rho}{\rho}$. We assume $0 < \alpha \ll 1$, $0 < \beta \ll 1$ and $\alpha = \lambda\beta$ where $\lambda \in (1, \infty)$. We consider the network delay model as in [16]. We then have $\gamma = \frac{\alpha}{1+\Delta\alpha}$ can be viewed as a "discounted" version of α due to the fact that the messages sent by honest parties can be delayed in Δ rounds; γ corresponds to the "effective" honest computing resource. We also assume $(\alpha + \beta)\Delta \ll 1$.

Theorem A.1 (Chain growth). *For any $\delta, \gamma > 0$, consider the blockchain protocol Π_{C} (see Section A.1) among a set $\mathcal{P}$ of players. For any honest player $P \in \mathcal{P}$ with the local chain C of length ℓ in round r and C' of length ℓ' in round r', where $r' - r = s > 0$, in $\textsc{View}_{\Pi_{\mathsf{C}}, \mathcal{A}, \mathcal{Z}}$, the probability that $\ell' - \ell \geq g \cdot s$ is at least $1 - e^{-\Omega(s)}$ where $g = (1-\delta)\gamma$.*

Theorem A.2 (Chain quality). *Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. For any $\delta > 0$, consider the blockchain protocol Π_{C} (see Section A.1) among a set $\mathcal{P}$ of players. For any honest player $P \in \mathcal{P}$, with the local chain C of length ℓ in $\textsc{View}_{\Pi_{\mathsf{C}}, \mathcal{A}, \mathcal{Z}}$, the probability that, for large enough T consecutive blocks of C which are generated in s rounds, the ratio of honest blocks is no less than $\mu = 1 - (1+\delta)\frac{\beta}{\gamma}$ is at least $1 - e^{-\Omega(T)}$.*

Theorem A.3 (Common prefix). *Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. For any $\delta > 0$, consider the blockchain protocol Π_{C} (see Section A.1) among a set $\mathcal{P}$ of players. For any two honest players, $P' \in \mathcal{P}$ with the local chain C' of length ℓ' in round r', and $P'' \in \mathcal{P}$ in round r'' with the local chain C'' of length ℓ'', in $\textsc{View}_{\Pi_{\mathsf{C}}, \mathcal{A}, \mathcal{Z}}$ where $r' \leq r''$, the probability that $\mathsf{C}'[1, l] \preceq \mathsf{C}''$ is at least $1 - e^{-\Omega(\kappa)}$ where $l = \ell' - \kappa$.*

Indeed, the underlying blockchain in the blockchain protocol is the header-chain. Thus, the three theorems above are essentially for the header-chain. In order words, the header-chains in the blockchain protocol Π_{C} satisfy the three properties.

B SUPPORTING MATERIAL FOR SECTION 5

B.1 Proof of Theorem 5.1

Proof. By Theorem A.3, the generalized blockchain protocol (the underlying header-chain) satisfies the common-prefix property with the probability $1 - e^{-\Omega(\kappa)}$. Now, suppose that the execution in FULL-mode Π^{FULL} (see Section 5.1) satisfies the snapshot persistence property, with probability at most $1 - \zeta(\kappa)$ such that $\zeta(\cdot)$ is a *non-negligible* function. We then show that there exists a ppt adversary $\mathcal{A}'$ against Σ_{au} that can win the collision-finding experiment (see Definition 3.2) with probability at least $\zeta'(\kappa)$, where $\zeta'(\cdot)$ is a *non-negligible* function.

Let $\mathcal{Z}$ denote the environment in which protocol execution is in the FULL-mode with n number of miners. We will construct $\mathcal{A}'$ from the execution of the FULL-mode Π^{FULL} that is directed by $\mathcal{Z}$ and run in $q(\kappa)$ rounds, where $q(\cdot)$ is a polynomial function, as follows. The adversary $\mathcal{A}'$ is given 1^κ. He then randomly chooses pair of honest players P_1 with the snapshot $\mathsf{SS}^1_{\ell_1} := \mathcal{T}^1_{\ell_1}$ of length ℓ_1 and P_2 with the snapshot $\mathsf{SS}^2_{\ell_2} := \mathcal{T}^2_{\ell_2}$ of length $\ell_2 = \ell_1 = \ell$, where $\mathcal{T}^1_{\ell_1} := \langle x^1_1, \ldots, x^1_{\ell_1} \rangle$ and $\mathcal{T}^2_{\ell_2} := \langle x^2_1, \ldots, x^2_{\ell_2} \rangle$ (note that, it holds that $\ell_2 \geq \ell_1$ when $r_2 \geq \Delta + r_1$). Then output a pair (x^1_ℓ, x^2_ℓ) to its challenger.

Note that, once τ_ℓ appears on the corresponding header-chain $\mathcal{H}^1_{\ell_1}$ of any player P_1 at round r_1, then for any other honest player P_2 with the corresponding header-chain $\mathcal{H}^2_{\ell_2}$ at round r_2, by the common-prefix property, τ_ℓ also appears on $\mathcal{H}^2_{\ell_2}$.Since the execution in FULL-mode Π^{FULL} (see Section 5), satisfies the snapshot persistence property, with probability at most $1 - \zeta(\kappa)$, then there exists a pair $(\mathcal{T}^i_{\ell_i}, \mathcal{T}^j_{\ell_j})$ of players (P_i, P_j) such that $\Pr[x^i_\ell \neq x^j_\ell] > \zeta$.

Thus,

$$\Pr[x^2_\ell \neq x^1_\ell] > \frac{\zeta}{n^2 q}$$

where $\frac{1}{n^2}$ is the probability that $P_1 = P_i$ and $P_2 = P_j$, and $\frac{1}{q}$ is the probability that ℓ is the length such that $x^2_\ell \neq x^1_\ell$, and q is a polynomial function. Note that, the transaction sets x^1_ℓ and x^2_ℓ have the same digest by the common-prefix property. Thus, the ppt adversary $\mathcal{A}'$ against Σ_{au} can win the collision-resistant experiment (see Definition 3.2) with non-negligible probability ζ' where $\zeta' = \frac{\zeta}{n^2 q}$. By the common-prefix property, we have $\mathcal{H}^1_\ell$, $\mathcal{H}^2_\ell$ (truncating the last κ headers) are the same with probability at least $1 - e^{-\Omega(\kappa)}$; it follows that $\tau^1_\ell = \tau^2_\ell$ (where $\tau^1_\ell := \mathsf{Root}(x^1_\ell)$ and $\tau^2_\ell := \mathsf{Root}(x^2_\ell)$) with probability that is close to 1. Thus, there exists a pair (x^1_ℓ, x^2_ℓ) such that $x^1_\ell \neq x^2_\ell$, $\mathsf{Root}(x^1_\ell) = \mathsf{Root}(x^2_\ell) = \tau^1_\ell = \tau^2_\ell$, $\mathsf{CheckRoot}(x^1_\ell, \tau^1_\ell) = \mathsf{CheckRoot}(x^2_\ell, \tau^2_\ell) = 1$, with probability at least ζ' where $\zeta' = \frac{\zeta}{n^2 q}$. □

B.2 Proof of Theorem 5.2

Proof. By Theorems A.2 and A.1, the generalized blockchain protocol (the underlying header-chain) satisfies the chain growth property and chain quality property with probability that is close to 1. Now, we need to prove that assuming all honest players receive as input the transaction `tx` for at most $t = (1+\delta)\frac{2\kappa}{\gamma}$ rounds, for $\delta > 0$, then for any player P there exists snapshots $\mathsf{SS}_\ell := \mathcal{T}_\ell = x_1||x_2||\ldots||x_\ell$ of length ℓ, $\mathsf{SS}_{\ell-1} := \mathcal{T}_{\ell-1}$ of length ℓ-1 and a transaction set x, in the FULL mode, such that $\mathtt{tx} \in x_\ell$, with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function and $\diamond := ||$. Note that, by the chain-growth property (Theorem A.1), we have that the header-chain of any honest party has increased by at least 2κ headers, respectively. By the chain-quality property (Theorem A.2), there exists a header with the digest of a transaction set x that was computed by an honest party after $t = (1+\delta)\frac{2\kappa}{\gamma}$ rounds, and $\mathtt{tx} \in x$, with probability that is close to 1.

Now, suppose the execution in FULL-mode achieves the snapshot liveness, with probability at most $1 - \zeta(\kappa)$, where $\zeta(\cdot)$ is a non-negligible function, then we need to show that there exists a ppt adversary $\mathcal{A}'$ against Σ_{au} that can win the collision-resistant experiment (see Definition 3.2) with probability at least $\zeta'(\kappa)$, where $\zeta'(\cdot)$ is a non-negligible function. Let $\mathcal{Z}$ denote the environment in which protocol execution is in the FULL-mode with n miners.

We then construct the adversary $\mathcal{A}'$ from the execution in the FULL-mode that is directed by $\mathcal{Z}$ and run in $q(\kappa)$ rounds, where $q(\cdot)$ is a polynomial function, as follows. The adversary $\mathcal{A}'$ is given 1^κ. Then, randomly choose a transaction set x that is input to the execution in a particular round r, with the corresponding digest τ. The adversary then randomly chooses a player P'. Let $\mathsf{SS}_{\ell} := \mathcal{T}_{\ell}$ be the snapshot of a player P' at round $r' \leq r + t$ where τ appears on the corresponding header-chain of $\mathcal{T}_{\ell}$, and Let $\mathsf{SS}_{\ell} := \mathcal{T}_{\ell+1}$ be the latest snapshot of a player P' of length ℓ+1 before round $r + t$. Let x' denote the corresponding transaction set such that $\mathsf{SS}_{\ell+1} := \mathsf{SS}_{\ell} \diamond x'$, where $\diamond := \|$. The adversary then outputs (x, x').

Since if the execution in FULL-mode Π^{FULL} (see Section 5.1) satisfies the snapshot liveness property, with probability at most $1 - \zeta(\kappa)$, then all transactions belong to a transaction set x such that x appears on $\mathcal{T}_{\ell}$ of any honest player P, with probability at most $1 - \zeta(\kappa)$. Thus, there exists a transaction set that is not on $\mathcal{T}_{\ell}'$ of an honest player P' with probability at least $\zeta(\kappa)$, i.e., $\Pr[\mathcal{T}_{\ell} \neq \mathcal{T}_{\ell}'] > \frac{\zeta}{n}$. It follows that

$$\Pr[x \neq x'] > \frac{\zeta}{nq}$$

where $\frac{1}{q}$ is the probability that $\mathtt{tx} \in x$ is the transaction that breaks the property (function q is a polynomial function) and $\frac{1}{n}$ is the probability that party P is P'. Therefore, there exists $x' \neq x$, with the same digest τ as x that appears on $\mathcal{T}_{\ell}$ of at least one player, with probability at least $\frac{\zeta}{nq}$.

Thus, the PPT adversary $\mathcal{A}'$ against Σ_{au} can win the collision-resistant experiment (see Definition 3.2) with probability at least $\zeta' = \frac{\zeta}{nq}$. By the chain growth property and chain quality property, the corresponding digest τ of x is definitely on a header in the header-chain of each honest player after $t = (1 + \delta)\frac{2\kappa}{\gamma}$ rounds, for $\delta > 0$. Thus, another transaction set $x' \neq x$, with the same digest such that $\mathsf{Root}(x) = \mathsf{Root}(x') = \tau$, $\mathsf{CheckRoot}(x', \tau) = 1$, replaces x, with probability at least $\zeta' = \frac{\zeta}{nq}$, where ζ is a non-negligible probability. Therefore, there exists a pair (x, x') such that $\mathsf{CheckRoot}(x, \tau) = \mathsf{CheckRoot}(x', \tau) = 1$, with a non-negligible probability. This completes the proof.

□

C SUPPORTING MATERIAL FOR SECTION 6

C.1 Proof of Lemma 6.1

PROOF. We need to prove that P_1 and P_2 have the same transaction set returned by operation ▲ (see Algorithm 1), with probability at least $1 - \epsilon(\kappa)$ under the assumptions that the authenticated data structure Σ_{au} (see Definition 3.1) is collision-resistant, and that $\gamma = \lambda\beta$ and $\lambda > 1$. By Theorem A.3, the generalized blockchain protocol (the underlying header-chain) satisfies the common-prefix property with probability at least $1 - e^{-\Omega(\kappa)}$. Now, suppose by contradiction that $x^{i}_{\ell-\kappa+1} = x^{j}_{\ell-\kappa+1}$, for all pairs of players (P_i, P_j), with probability at most $1 - \zeta(\kappa)$ where $\zeta(\cdot)$ is a non-negligible function, then we need to show there exists a PPT adversary $\mathcal{A}'$ that can win the collision-resistant experiment with probability at least $\zeta'(\kappa)$ where $\zeta'(\cdot)$ is a non-negligible function.

We construct the adversary $\mathcal{A}'$ as follows. The adversary $\mathcal{A}'$ upon receiving input 1^{κ} from its challenger, runs the execution such that all players have the same UTXO-sets, then when the UTXO-sets are extended to the length $\ell - \kappa + 1$, let all honest players change to PRUNE mode. Let x_i denote the new transaction set that is injected when player P_i changes to PRUNE mode, for $i \in [n]$ (here, n is the number of players in the system). Then randomly choose a pair of player (P_1, P_2), and output the pair of transaction sets $(x^{1}_{\ell-\kappa+1}, x^{2}_{\ell-\kappa+1})$.

Since we have $x^{i}_{\ell-\kappa+1} = x^{j}_{\ell-\kappa+1}$, for all pairs (P_i, P_j), with probability at most $1 - \zeta(\kappa)$. This implies that there exists a pair (P_i, P_j) such that $\Pr[x^{i}_{\ell-\kappa+1} \neq x^{j}_{\ell-\kappa+1}] > \zeta$ Thus, $\Pr[x^{1}_{\ell-\kappa+1} \neq x^{2}_{\ell-\kappa+1}] > \frac{\zeta}{n^2}$ where $\frac{1}{n^2}$ is the probability that $P_1 = P_i$ and $P_2 = P_j$. Thus, the adversary $\mathcal{A}'$ that can win the collision-resistant experiment with a non-negligible probability $\zeta' = \frac{\zeta}{n^2}$. Therefore, we conclude that $x^{1}_{\ell-\kappa+1} = x^{2}_{\ell-\kappa+1}$ with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function. It follows that, with probability at least $1 - \epsilon(\kappa)$

$$\mathcal{U}^{1}_{\ell-\kappa+1} = \mathcal{U}^{1}_{\ell-\kappa} \circ x^{1}_{\ell-\kappa+1} = \mathcal{U}^{2}_{\ell-\kappa} \circ x^{2}_{\ell-\kappa+1} = \mathcal{U}^{2}_{\ell-\kappa+1}$$

□

C.2 Proof of Theorem 6.2

PROOF. We prove by induction as follows. Initially, we have $\mathsf{SS}^{1}_{\emptyset} = \mathsf{SS}^{2}_{\emptyset} = \mathcal{U}_0$. Assume it holds for length ℓ, where P_1 and P_2 are both in PRUNE mode that $\mathcal{U}^{1}_{\ell-\kappa} = \mathcal{U}^{2}_{\ell-\kappa}$ with probability at least $1 - \epsilon(\kappa)$. From Lemma 6.1, it holds that $\mathcal{U}^{1}_{\ell-\kappa+1} = \mathcal{U}^{2}_{\ell-\kappa+1}$ with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.

□

C.3 Proof of Theorem 6.3

PROOF. The proof for this theorem is similar to the proof for Theorem B.2. However, since the adversary cannot derive a transaction set from a UTXO set, he would output his UTXO set with a UTXO set of an honest player in the experiment. □

D SUPPORTING MATERIAL FOR SECTION 7

D.1 Proof of Lemma 7.1

PROOF. We show that

$$\mathbb{V}_{\text{FULL}}(\mathsf{SS}^{\mathsf{f}}_{\ell},\ x) = \mathbb{V}_{\text{PRUNE}}(\mathsf{SS}^{\mathsf{p}}_{\ell}, x) \tag{1}$$

where $\mathsf{SS}^{\mathsf{f}}_{\ell} := \mathcal{T}_{\ell}$ and $\mathsf{SS}^{\mathsf{p}}_{\ell} := \mathcal{U}_{\ell-\kappa}$. Now, consider two cases:
Case 1: If the transaction inputs of all valid transactions $\mathtt{tx}$ in x are in $\mathcal{U}_{\ell-\kappa}$, then by definition, the data validation predicate in PRUNE mode $\mathbb{V}_{\text{PRUNE}}$ (See Table 2) will return true. In addition, the transaction inputs are from the unspent transaction outputs, those transactions in x will not conflict with the ones in $\mathcal{T}_{\ell}$, by definition, predicate $\mathbb{V}_{\text{FULL}}$ (see Table 1) will return true.
Case 2: If the transaction inputs of any valid transactions $\mathtt{tx}$ in x are not in $\mathcal{U}_{\ell-\kappa}$, then the data validation predicate in PRUNE mode $\mathbb{V}_{\text{PRUNE}}$ will return false. In this case, the transaction outputs might be originated from spent transaction outputs in $\mathcal{T}_{\ell}$ or not from any outputs in $\mathcal{T}_{\ell}$. In the former case where they are from the spent transaction outputs, $\mathsf{NonConflict}(\cdot)$ will return 0. In the later case where they are not from any output, $\mathsf{Traceable}(\cdot)$ will return 0. In any cases, the predicate $\mathbb{V}_{\text{FULL}}$ will return false.

□

D.2 Proof of Theorem 7.2

Proof. Consider any honest player P_1 having the generalized chain (with the corresponding snapshot) of length ℓ_1 at round r_1 and honest player P_2 having the generalized chain (with the corresponding snapshot) of length ℓ_2 at round r_2. By the common-prefix property of the generalized blockchain protocol (the underlying header-chain), the header-chain of P_2 will be at least as long the header-chain of P_1 at round $r_2 \geq r_1 + \Delta$, i.e., $\ell_2 \geq \ell_1$. Consider that $\ell_2 = \ell_1 = \ell$, we then have the following important cases:

Case 1: If P_1 and P_2 are both in FULL mode. Here, the snapshot of P_1 in FULL mode $\mathsf{SS}_{\ell}^{\mathsf{f},1} := \mathcal{T}_{\ell}^{1}$ and the snapshot of P_2 in FULL mode $\mathsf{SS}_{\ell}^{\mathsf{f},2} := \mathcal{T}_{\ell}^{2}$. By Theorem 5.1, we have that $\mathcal{T}_{\ell}^{2} = \mathcal{T}_{\ell}^{1}$, with probability at least $1-\epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function. Thus, for any transaction set x, it follows that

$$\mathbb{V}_{\textsc{full}}(\mathsf{SS}_{\ell}^{\mathsf{f},1},\ x) = \mathbb{V}_{\textsc{full}}(\mathsf{SS}_{\ell}^{\mathsf{f},2}, x)$$

with probability at least $1-\epsilon(\kappa)$.

Case 2: If both P_1 and P_2 are in PRUNE mode. Here, the snapshot of P_1 in PRUNE mode $\mathsf{SS}_{\ell}^{\mathsf{p},1} := \mathcal{U}_{\ell-\kappa}^{1}$ and the snapshot of P_2 in PRUNE mode $\mathsf{SS}_{\ell}^{\mathsf{p},2} := \mathcal{U}_{\ell-\kappa}^{2}$. By Theorem 6.2, we have $\mathcal{U}_{\ell-\kappa}^{1} = \mathcal{U}_{\ell-\kappa}^{2}$. Thus, for any transaction set x, it follows that

$$\mathbb{V}_{\textsc{prune}}(\mathsf{SS}_{\ell}^{\mathsf{p},1},\ x) = \mathbb{V}_{\textsc{prune}}(\mathsf{SS}_{\ell}^{\mathsf{p},2}, x)$$

with probability at least $1-\epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function.

Case 3: If P_1 or P_2 is not in the FULL mode. Consider player P_1 in the FULL mode and P_2 in PRUNE mode. Initially, we have $\mathsf{SS}_{\emptyset}^{\mathsf{f},1} = \epsilon$ and $\mathsf{SS}_{\emptyset}^{\mathsf{p},1} = \mathcal{U}_0$. By definitions of the predicates $\mathbb{V}_{\textsc{full}}(\cdot)$ and $\mathbb{V}_{\textsc{prune}}(\cdot')$, and the assumption that the initial UTXO set is valid with respect to the initialization of the ledger in the FULL-mode (see Section 6.2), it holds that $\mathbb{V}_{\textsc{full}}(\mathsf{SS}_{\emptyset}^{\mathsf{f},1}) = \mathbb{V}_{\textsc{prune}}(\mathsf{SS}_{\emptyset}^{\mathsf{p},2}) = 1$, where $\mathsf{SS}_{\emptyset}^{\mathsf{f},1} = \epsilon$ and $\mathsf{SS}_{\emptyset}^{\mathsf{p}} = \mathcal{U}_0$ denote the initial snapshots in PRUNE and FULL modes, respectively.

By the full mode snapshot persistence (Theorem 5.1), it holds that, for any pair of (P_1, P_2), we have $x_i^1 = x_i^1$, for all $i \in [\ell - \kappa]$, with probability at least $1-\epsilon(\kappa)$. We also have the snapshot of P_1 in FULL mode $\mathsf{SS}_{\ell}^{\mathsf{f},1} := \mathcal{T}_{\ell}^{1} = \langle x_1^1, \ldots, x_{\ell-\kappa}^1 \rangle$, and the snapshot of P_2 in PRUNE mode $\mathsf{SS}_{\ell}^{\mathsf{p},2} := \mathcal{U}_{\ell-\kappa}^{2} = \mathcal{U}_0 \circ x_1^2 \circ \ldots \circ x_{\ell-\kappa}^2$. From Lemma 7.1, we conclude that, for any transaction set x, with probability at least $1-\epsilon(\kappa)$,

$$\mathbb{V}_{\textsc{full}}(\mathsf{SS}_{\ell}^{\mathsf{f},1},\ x) = \mathbb{V}_{\textsc{prune}}(\mathsf{SS}_{\ell}^{\mathsf{p},2}, x)$$

□