Multi-mode Cryptocurrency Systems

Tuyet Duong
VCU

Hong-Sheng Zhou
VCU

Alexander Chepurnoy
Ergo and IOHK

October 5, 2017

Abstract

Bitcoin is a decentralized digital cash system on top of a distributed, append-only public ledger. Bitcoin transactions are recorded in the public ledger, and the ledger is maintained by a network of nodes via proof-of-work mechanism. Among these network nodes, some are in "full mode", and the remaining are in certain "light" modes. Each full-mode node records the complete history of transactions, which allows the node to verify the validity of new transactions by itself. The nodes in a light mode may only store a partial history of transactions, or a succinct version of the history; this allows the Bitcoin protocol to be executed in computing devices where storage is limited.

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 protocols, i.e., the protocol running among full-mode nodes. In this paper, we initial the study of *multi-mode* cryptocurrency protocols. We generalize the recent framework by Garay et al (Eurocrypt 2015) with new security definitions, so that the security of a larger class of protocols can be analyzed.

As an immediate application of our new framework, we analyze the security of existing blockchain pruning proposals for Bitcoin and Ethereum aiming to improve the storage efficiency of network nodes by pruning unnecessary information from the ledger.

1 Introduction

The rise of blockchain technologies. Bitcoin is a decentralized digital cash system on top of a distributed, append-only public transaction ledger. The public ledger is maintained by a network of nodes via the so-called 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 crowd funding, decentralized automatous organization, come to the reality.

Multi-mode cryptocurrency, and the advantage of having multiple modes. 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; 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 [14], 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 [24] is also a multi-mode system. For example, Parity clients [22] for Ethereum blockchain can be run in a light mode called "WarpSync" mode [23].

The advantage of having multi 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; 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 will allow more players especially those with limited storage to join the system. This apparently will make multi-mode Bitcoin like cryptocurrencies more popular.

Understanding the security of cryptocurrency systems. Rigorous security analysis of Bitcoin protocols has been recently developed. Garay et al. [7] proposed the first cryptographic framework and investigated the security of a *simplified* version of Bitcoin. More concretely, Garay et al investigated a single mode version of Bitcoin protocol where all nodes

[1] Please see Section 8 of the whitepaper [14] for details.

are in the full-mode[2], 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 [17, 8]. 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

Technical challenges. To answer the above question, we first need to understand the technical difficulty that we are facing. First, we need a *stateful* model supporting *different* roles; each player is expected to maintain a local memory for storing the complete or partial history of transactions. Secondly, we need to define security properties for the light modes. In the previous efforts [7, 17], 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. Finally, we need to define security for multiple modes. It is very possible that, nodes in different modes may not be able to work together. If that is the case, the entire system may be insecure.

Our approach. We consider a stateful model where each player maintains his local memory to store and update his version of the transaction history. We consider different types of players including full-mode players who store the complete history, and light-mode players who store a compressed version of the history. To address the second challenge, we introduce *snapshot* by generalizing the notion of ledger properly. Based on the notion of snapshot, we define the relaxed version of persistence and liveness. Finally, to address the third challenge, we define the *soundness* property to ensure the nodes in different modes are compatible. Intuitively, for each new transaction, nodes in the full mode and nodes in the light modes should return the same answer: all of these nodes may accept the new transaction, or reject the transaction.

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: 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 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.

Related work: Due to space limitation, the related work can be found in Appendix A.1.

Organization: We introduce the model in Section 2. Then we introduce the proof of work blockchain in Section 3; due to space limitation, additional materials can be found in Appendix B.

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 appendix.

2 Model

In order to study the security of Bitcoin-like protocols, Garay et al. [7] (then Pass et al [17]) proposed a cryptographic framework and showed that the (simplified) Bitcoin can achieve several important security properties. Based on the previous modeling efforts [7, 17], we consider the 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.[3]

[2] We remark that, Garay et al actually investigated an even further simplified version of Bitcoin, called Bitcoin Backbone, where transactions are straightforwardly recorded in each block of blockchain. In the real world Bitcoin blockchain, the transactions are organized via authenticated data structure. Please see Section 5 for details.

[3] We note that, in [7], only a stateless model is considered; there, each player is note expected to maintain a local memory for storing the complete or partial history of transactions. Instead, players are allowed to obtain the entire blockchain from the network. This modeling choice is sufficient for the security analysis of a single mode Bitcoin blockchain where all players are in full-mode. However, the modeling does not reflect the reality and it is not sufficient for analyzing blockchain protocols with light mode players. Pass et al [17] considered a *stateful* model, for the single mode Bitcoin system. The model here is very close to that in [17], but for multi-mode blockchain systems.

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 channels 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.

Creation phase. 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, after the preparation phase can properly join the mining process. 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.

Deletion phase. Active miners can be deleted from the system. In this case, the miners will be set inactive. These inactive miners are not allowed to receive any input from, or return any output to the environment, and they are not allowed to send out messages to Broadcast.

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 $\{\text{View}^P_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ 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 $\text{View}^P_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$. The concatenation of the view of all miners $\langle \text{View}^P_{\Pi,\mathcal{A},\mathcal{Z}} \rangle_{P\in\mathcal{P}}$ is denoted by $\text{View}_{\Pi,\mathcal{A},\mathcal{Z}}$.

3 Proof-of-work blockchain

In this section we provide definitions for a proof-of-work blockchain. The definitions are similar to previous works [7, 17]. 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 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 $h_j = hash(h_{j-1})$ 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 $\pi_{\mathtt{au}}$, see Definition C.1 further), and w_j is a proof-of-work puzzle solution. The header should satisfy the inequality $hash(\mathsf{head}_j) < \mathtt{T}$, where $\mathtt{T}$ is the proof-of-work target setting hardness of a proof-of-work puzzle solution.

A *blockchain* is a sequence of ordered blocks, $\mathcal{B}_\emptyset, \mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$, which are probably linked, so for any block B_j, its header contains correct link $h_j = hash(h_{j-1})$, with the exception of $h_1 = 0$. We denote the blockchain as $\mathcal{C} = \mathcal{B}_\emptyset || \mathcal{B}_1 || \mathcal{B}_2 || \ldots || \mathcal{B}_\ell$, where operation "||" indicates the concatenation between any two blocks.

Blockchain payload and payload validation predicate. A *blockchain payload* is concatenation of blockchain block payloads. In details, for blockchain $\mathcal{C}$, blockchain payload is $x_\mathcal{C} = \langle x_1, \ldots, x_\ell \rangle$. 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 3.1 (Payload validation predicate)**.** *Consider a blockchain $\mathcal{C}$ of length $\ell \in \mathbb{N}$, with payload $x_\mathcal{C} = \langle x_1, \ldots, x_\ell \rangle$. We say deterministic $\mathbf{V}(\cdot)$ is a* payload validation predicate *for blockchain $\mathcal{C}$, if all the following conditions hold (1) $\mathbf{V}(\epsilon) = 1$, (2) if $\mathbf{V}(x_\mathcal{C}) = 1$, then there exists a new payload x such that $\mathbf{V}(x_\mathcal{C}, x) = 1$,*

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 next sections, we will instantiate predicate $\mathbf{V}(\cdot)$ when we consider a concrete blockchain application.

3.2 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 3.2 (Stable and unstable blockchain payload)**.** *Let $\mathsf{B}_1, \ldots, \mathsf{B}_\ell$ 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 $j \in [\ell]$. 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, $\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_\ell$. However, different from the simplified proof-of-work based blockchain that we present in the previous subsection (where the blockchain is viewed as the concatenation of blocks), 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 blockchain 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 $\mathsf{C} \preceq \mathsf{C}'$ (where $\mathsf{C}' := \langle \mathcal{H}'_\ell, \bar{x}'_\mathsf{C}, \tilde{x}'_\mathsf{C} \rangle$) to say $\mathcal{H}_\ell \preceq \mathcal{H}'_\ell$.

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 $\mathcal{B}_{\ell+1}$ 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 ▲.

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

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 *useful* information, it can be instantiated as the operation $\circ$ (see Section D.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 depending on the applications.

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 3.3 (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 $\mathsf{V}(\bar{\mathbf{x}}_{\mathsf{C}} \diamond x) = 1$*

3.3 Security properties for the blockchain

In [7] (and then [17, 10], several important security properties, *common prefix property* [7, 17], *chain quality property* [7], and *chain growth property* [10], have been defined for (simplified) Bitcoin blockchain protocols.

Definition 3.4 (Chain growth)**.** *Consider a blockchain protocol $\Pi_{\mathcal{C}}$ among a set $\mathcal{P}$ of players. The chain growth property $\mathcal{Q}_{\text{cg}}$ 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 $\text{VIEW}_{\Pi_{\mathcal{C}},\mathcal{A},\mathcal{Z}}$. It holds that $\ell' - \ell \geq g \cdot (r' - r)$ for $g > 0$.*

Definition 3.5 (Chain quality)**.** *Consider a blockchain protocol $\Pi_{\mathcal{C}}$ among a set $\mathcal{P}$ of players. The chain quality property $\mathcal{Q}_{\text{cq}}$ 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 $\text{VIEW}_{\Pi_{\mathcal{C}},\mathcal{A},\mathcal{Z}}$, it holds that for large enough T consecutive blocks of C the ratio of honest headers is at least μ.*

Definition 3.6 (Common prefix)**.** *Consider a blockchain protocol $\Pi_{\mathcal{C}}$ among a set $\mathcal{P}$ of players. The common prefix property $\mathcal{Q}_{\text{cp}}$ 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 $\text{VIEW}_{\Pi_{\mathcal{C}},\mathcal{A},\mathcal{Z}}$ where $P', P'' \in \mathcal{P}$, $r' \leq r''$, it holds that $\mathsf{C}'[1, l] \preceq \mathsf{C}''$ 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: Ledger

Transactions. Generalizing a transaction structure of the Bitcoin and its descendants, we assume that a transaction $\mathtt{tx}$ is associated with (1) *transaction inputs*, where a transaction input is denoted by Inp, and (2) also new *transaction outputs* to create, where a transaction output is denoted by Outp. Here, an input or output is considered as a bitstring, i.e., $\mathsf{Inp} \in \{0,1\}^*$, $\mathsf{Outp} \in \{0,1\}^*$, and $\mathtt{tx} := (\langle \mathsf{Inp}_1, \ldots, \mathsf{Inp}_j\rangle, \langle \mathsf{Outp}_1, \ldots, \mathsf{Outp}_k\rangle)$, for $j, k \in \mathbb{N}$. Informally, a

transaction $\mathtt{tx}$ is valid if it spends outputs created by previous transactions (or exist before the first transaction) and not spent yet by other transactions, and also some application-specific checks are to be applied (such as signature or amount validation).

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 $\mathcal{C}$ 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 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 4.1 (Ledger)**.** *Let κ be the security parameter. Consider a chain $\mathcal{C}$ of the length $\ell \in \mathbb{N}$ which contains a blockchain payload $x_{\mathcal{C}} := \langle x_1, \ldots, x_\ell \rangle$ We say $\mathcal{L}_\ell$ is the ledger (of length ℓ) of $\mathcal{C}$, 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., $x_{\mathcal{C}} := \langle x_1, \ldots, x_{\ell-\kappa} \rangle$, where x_i is the input of the i-th block in $\mathcal{C}$ and $x_i := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$. In addition, it should hold that $\mathbf{V}(\mathcal{L}_\ell) = 1$, where $\mathbf{V}(\cdot)$ is a payload validation predicate (Definition 3.1).

Remark 4.2. *Here we follow the work by Pass et al. [17], 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.*

4.2 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, $x := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$. However, from the generalized chain C of the length ℓ, we obtain a generalized ledger, called *snapshot* and denoted $\mathtt{SS}_\ell$. 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., $\mathtt{SS}_\ell = x_1 \diamond \ldots \diamond x_{\ell-\kappa}$, where $x_i := \langle \mathtt{tx}_1, \mathtt{tx}_2, \ldots, \mathtt{tx}_e \rangle$ is the input. In other words, the snapshot $\mathtt{SS}_\ell$ is the compressed stable blockchain payload. Moreover, it should hold that $\mathsf{V}(\mathtt{SS}_\ell) = 1$, where $\mathsf{V}(\cdot)$ is a compressed payload validation predicate (Definition 3.3). We formally define a snapshot as follows.

Definition 4.3 (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 $\mathtt{SS}_\ell$ is the snapshot (of length ℓ) of $\mathcal{C}$, if $\mathtt{SS}_\ell = \bar{\mathbf{x}}_{\mathsf{C}}$.*

Generalized operation rules. The snapshot $\mathtt{SS}_\ell$ 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 4.4 (Generalized operation rule)**.** *Let $\mathsf{B}_1, \ldots, \mathsf{B}_\ell$ be the ordered sequence of blocks. Consider a generalized blockchain C of the length $\ell \in \mathbb{N}$, with the corresponding snapshot $\mathtt{SS}_\ell$ 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) $\mathtt{SS}_\ell = x_1 \diamond \ldots \diamond x_{\ell-\kappa}$ where x_i is the payload of block B_i, and (2) $\mathsf{V}(\mathtt{SS}_\ell) = 1$.*

Security properties for snapshot. There are two important security properties, *snapshot persistence* and *snapshot liveness*, in the single-mode generalized ledger protocol. Here we investigate how to define the relaxed security properties for the generalized ledger protocol when nodes are running in the snapshot mode.

Snapshot-persistence. Intuitively, the snapshot-persistence property implies, during a time period, the recorded information (i.e., the compressed version of transactions) on two different ledgers (two players) of any pair of honest parties should be consistent with each other. Formally, we state the definition as follows.

Definition 4.5 (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 $\mathtt{SS}_{\ell_1}$ be the resulting snapshot at round r_1 of the length ℓ_1 reported by player P_1, where $\ell_1 > \kappa$.*

Here snapshot persistence property states that, for any player P_2, there exists a round r_2 where $r_2 \geq r_1 + \Delta$ such that if the reported snapshot $\mathtt{SS}_{\ell_2}$ (by player P_2) of the length ℓ_2 satisfies $\ell_2 = \ell_1$, then $\mathtt{SS}_{\ell_2} = \mathtt{SS}_{\ell_1}$.

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 4.6 (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 $\mathtt{SS}_\ell, \mathtt{SS}_{\ell-1}$ and a transaction set x (reported by P) of the length $\ell > \kappa$, in round $r' \leq r + t$, such that $\mathtt{SS}_\ell := \mathtt{SS}_{\ell-1} \diamond x$ and $\mathtt{tx} \in x$.

4.3 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 4.7 (Multi-mode snapshot validation predicate)**.** *Consider a multi-mode ledger protocol Π^{multi} with m modes $\mathtt{M}_1, \mathtt{M}_2, \ldots, \mathtt{M}_m$. For each mode $\mathtt{M}_i$ with operation rule $(\diamond_i, \mathbb{V}_i(\cdot))$, the corresponding initial snapshot is $\mathtt{SS}_\emptyset^{\mathtt{M}_i}$. Consider a generalized blockchain C of length ℓ, with the corresponding snapshot $\mathtt{SS}_\ell^{\mathtt{M}_i}$, 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 $\mathtt{M}_i$, where $i \in [m]$, it holds that $\mathbb{V}_i(\mathtt{SS}_\emptyset^{\mathtt{M}_i}) = 1$.*
- *for any mode $\mathtt{M}_i$, where $i \in [m]$, it holds that, if $\mathbb{V}_i(\mathtt{SS}_\ell^{\mathtt{M}_i}) = 1$, then there exists a new payload x such that $\mathbb{V}_i(\mathtt{SS}_\ell^{\mathtt{M}_i} \diamond_i x) = 1$,*
- *for any pair of modes $\mathtt{M}_i$ and $\mathtt{M}_j$, where $i, j \in [m]$, it holds that $\mathbb{V}_i(\mathtt{SS}_\ell^{\mathtt{M}_i}) = \mathbb{V}_j(\mathtt{SS}_\ell^{\mathtt{M}_j})$.*

We illustrate the multi-mode design in Figure 1: the 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 $(\diamond_i, \mathbb{V}_i(\cdot))$.

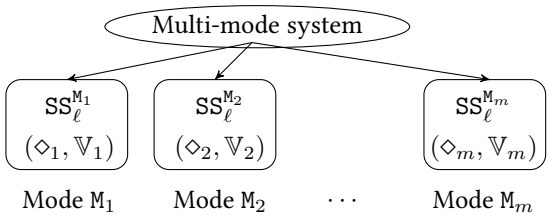

Figure 1: Multi-mode cryptocurrency system.

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 4.8 (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 $\mathtt{SS}_{\ell_1}^{\mathtt{M}_i}$ be the snapshot of player P_1 in the mode i-th of the length $\ell_1 > \kappa$ at round r_1.*

Here multi-mode soundness property states that, for any player P_2 in the mode j-th with resulting snapshot $\mathtt{SS}_{\ell_2}^{\mathtt{M}_j}$ at round $r_2 \geq r_1 + \Delta$ of the length ℓ_2, if $\ell_1 = \ell_2 = \ell$, then $\mathbb{V}_i(\mathtt{SS}_{\ell}^{\mathtt{M}_i} \diamond_i x) = \mathbb{V}_j(\mathtt{SS}_{\ell}^{\mathtt{M}_j} \diamond_j x)$, 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 2 for an illustration of Table 1.

Table 1: Snapshot Instantiation: full mode Π^{full}

$\mathtt{SS}_\ell$	$\mathcal{T}_\ell$ 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 $\mathcal{T}_\ell = \langle x_1, x_2, \ldots, x_{\ell-\kappa} \rangle$, – $\mathbb{V}_{\text{FULL}}$ outputs true if and only if the vector $\langle x_1, \ldots, x_{\ell-\kappa} \rangle$ is a valid sequence of transaction sets, i.e., for every transaction of the sequence $\mathtt{tx}_i := (\langle \mathsf{Inp}_1, \ldots, \mathsf{Inp}_j \rangle, \langle \mathsf{Outp}_1, \ldots, \mathsf{Outp}_k \rangle)$, for $j, k \in \mathbb{N}$, each input Inp spends an output created but not spent before $\mathtt{tx}_i$, and $\mathtt{tx}_i$ is correct from an application's point of view.

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 $\mathtt{SS}_{\ell+1} := \mathtt{SS}_\ell \diamond x$ and $\mathbb{V}_{\text{FULL}}(\mathtt{SS}_{\ell+1})$, 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.

Remark 5.1. *We emphasize that, in order to simplify the presentation, we consider that the snapshot is formed by the transactions (without considering the digests). Furthermore, we only focus on the conflict between transactions, and assume that the digests are generated correctly.*

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 4.3, Definition 4.5). 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.

Theorem 5.2 (full-mode persistence). *Consider the generalized ledger protocol Π^{full} (see Section 5.1). Let κ be the security parameter. Assume the authenticated data structure $\Sigma_{\mathtt{au}}$ (see Definition C.1) is collision-resistant. Assume that $\gamma = \lambda\beta$*

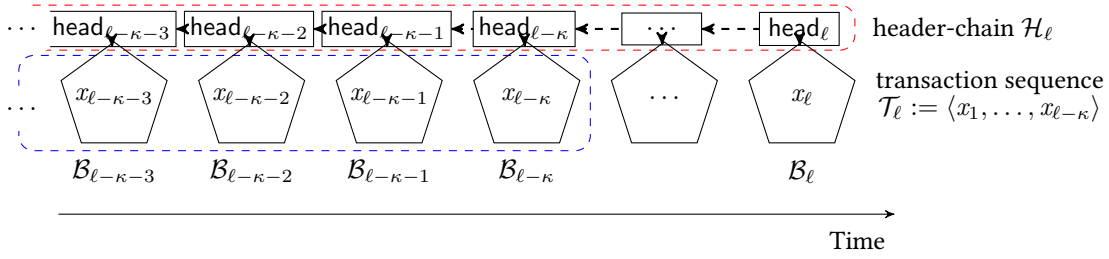

Figure 2: Structure of the generalized blockchain and ledger in full mode.

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 C.2.

Note that snapshot persistence property (see Section 4.3, Definition 4.6) 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 5.3. Proof can be found in Appendix C.3.

Theorem 5.3 (full-mode liveness)**.** *Consider the generalized ledger protocol Π^{full} (see Section 5.1). Let κ be the security parameter. Assume the authenticated data structure $\Sigma_{\mathtt{au}}$ (see Definition C.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 Instantiating snapshot to prune mode

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 $\{\ell\text{-}\kappa\}$-th UTXO-set. In Table 2, we instantiate the snapshot $\mathtt{SS}_\ell$ of any player P. (We write $\mathtt{SS}_\ell^j$, if we are interested in the snapshot of a particular player P_j, where $1 \leq j \leq n$.)

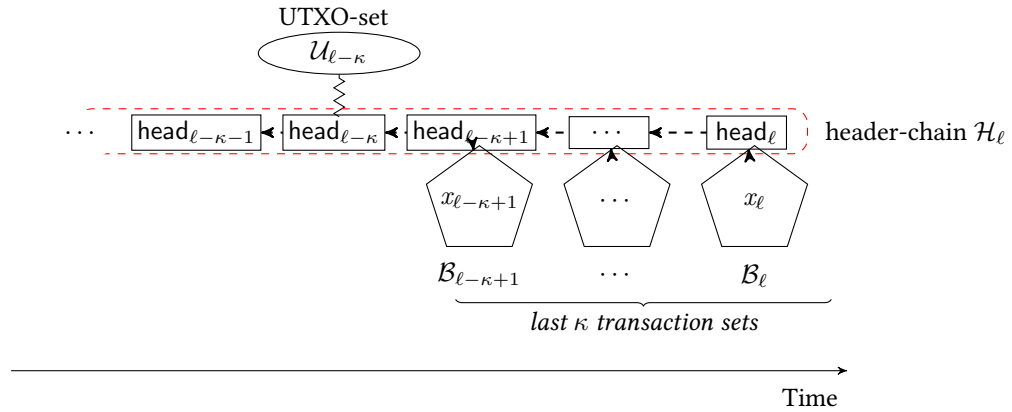

Figure 3: The illustration of the generalized blockchain and the ledger in prune mode. Here, the snapshot $\mathtt{SS}_\ell$ consists of the $(\ell - \kappa)$-th UTXO-set $\mathcal{U}_{\ell-\kappa}$.

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 ▲, see Algorithm 1), it should follow that $\mathtt{SS}_{\ell+1} := \mathtt{SS}_\ell \diamond x$ and $\mathbb{V}_{\text{FULL}}(\mathtt{SS}_{\ell+1})$, where $\diamond := \circ$. This implies, $\mathcal{U}_{\ell-\kappa+1} := \mathcal{U}_{\ell-\kappa} \circ x$.

Table 2: Snapshot instantiation: prune mode Π^{prune}

SS_ℓ	$\mathcal{U}_{\ell-\kappa}$ where $\mathcal{U}_{\ell-\kappa}$ is the $(\ell\text{-}\kappa)$-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 $(\mathcal{U}_{\ell-\kappa})$: – $\mathbb{V}_{\text{PRUNE}}$ returns true if $\mathcal{U}_{\ell-\kappa}$ can be parsed as a sequence of transaction outputs $\langle \mathsf{Outp}_1, \ldots, \mathsf{Outp}_k \rangle$, for $k \in \mathbb{N}$ that is correct from an application's point of view (for example, for a cryptocurrency sum of output values must equal to number of tokens issued to the moment).

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.

6.2 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 4.3, Definitions 4.5 and 4.6), 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 D.2.)

Lemma 6.1. *Consider the generalized ledger protocol Π^{prune} (see Section 6.1). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition C.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 generalized ledger protocol Π^{prune} (see Section 6.1). Let κ be the security parameter. Assume the authenticated data structure Σ_{au} (see Definition C.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 D.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 $\Pi^{\text{multi}} = (\Pi^{\text{full}}, \Pi^{\text{prune}})$. 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 [23].

7.1 Security analysis for incorporating full-mode and prune-mode

In this section, we prove the multi-mode system soundness (see Section 4.3, Definition 4.8). 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* $\Pi^{\text{multi}} = (\Pi^{prune}, \Pi^{full})$. *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 E.1.

Armed with Lemma 7.1 and Lemma 6.1 in Section 6.2, we are now ready to prove our main theorem as follows.

Theorem 7.2 (Multi-mode soundness)**.** *Consider* PRUNE *system* $\Pi^{\text{multi}} = (\Pi^{prune}, \Pi^{full})$ *in Section 7. Let* κ *be the security parameter. Let* κ *be the security parameter. Assume the authenticated data structure* π_{au} *(see Definition C.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 E.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] Some miners generating invalid blocks. https://bitcoin.org/en/alert/2015-07-04-spv-mining.

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

[5] 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.

[6] Davide Frey, Marc X Makkes, Pierre-Louis Roman, François Taïani, and Spyros Voulgaris. Bringing secure bitcoin transactions to your smartphone. In *Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware*, page 3. ACM, 2016.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

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

[15] 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.

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

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

[18] Leonid Reyzin, Dmitry Meshkov, Alexander Chepurnoy, and Sasha Ivanov. Improving authenticated dynamic dictionaries, with applications to cryptocurrencies. In *FC*, 2017. http://eprint.iacr.org/2016/994.

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

[20] 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.

[21] 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.

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

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

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

A Supporting material for Section 1

A.1 Related work

Cryptocurrency client modes. Multi-mode paradigm was introduced in the original Bitcoin whitepaper [14]: Nakamoto describes how to implement a "full" mode node; in addition, he also introduces a light mode called "SPV (Simple Payment Verification)" mode (please 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. An SPV node works under an assumption that a chain with most work contributed is also a valid chain. Thus an SPV client is not downloading and verifying transactions in blocks, but just checks validity of linking structure of the blockchain and also proofs of work by using only compact block headers. SPV could be problematic from security point of view, especially when miners are working in SPV mode; for example, miners may generate invalid blocks (see [3] for more details). A hybrid approach of secure thin client was proposed in [6]. A secure thin client is doing adaptive verification of blocks with regards to the level of targeted confidence. That is, this kind of client is checking part of transactions in arbitrarily-chosen suffix of the blockchain, and a user is choosing how much resources to spend on this probabilistic validation (more resources to be spent means higher confidence in the suffix contents).

Since version 0.11 of Bitcoin Core [1], a node can be run in another light 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 [24] has representation of state needed for transaction validation being fixed by the protocol, with an authenticating digest to be included into a block. Some implementations of the protocol then are using the digest to obtain a verifiable state snapshot from peers without building it by processing blocks from genesis. WarpSync [16] mode in Parity [22] is the example of this bootstrapping procedure. WarpSync 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 then is that there are network nodes storing authenticated validation states buried deep enough in the history. The scheme lacks rigorous security analysis.

There are some informal proposals in the community, as well as the paper [18], which propose to avoid storing validation state for non-mining full nodes, while miners are still storing this structure. In this proposal [18] miners are including proofs of state transformations against a root digest of two-party-model dynamic authenticated dictionary built on top of the state. It is possible for a full node then to check proofs and get a new digest value from an older one and a set of transformations, with some genesis digest all the protocol participants are agree on.

Cryptocurrency and security analysis. The security of Bitcoin system has been analyzed in the rational setting, e.g., [5, 4, 15, 9, 19, 20], and also in cryptographic setting [7, 17, 21, 10, 11]. Three important security properties, *common prefix*, *chain growth*, and *chain quality*, have been considered for secure blockchain protocols. The common prefix and chain quality properties were originally formalized by Garay et al. [7]. The chain growth property was first formally defined by Kiayias et al. [10]. The common prefix property was later strengthened by Pass et al. [17]. In our study, we adopt the stronger variant of the common prefix property by Pass et al. [17] together with the chain quality and chain growth from [10, 7].

B Supporting material for Section 3

B.1 The execution of miners

The blockchain protocol $\Pi_{\mathcal{C}}$. We are now ready to describe the blockchain protocol $\Pi_{\mathcal{C}}$ (Algorithm 2). This is the protocol that is executed by the miners and which is assumed to run "indefinitely".

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 5). 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 3) which is issuing a single query to idealized hash function per round. The pseudocode of main miner loop is provided in Algorithm 2.

Algorithm 2 The blockchain protocol $\Pi_{\mathcal{C}}$.

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

Algorithm 3 The proof of work function Pow, parametrized by T and hash function $hash(\cdot)$.

function $Pow(\mathsf{head}_\ell, \tau)$
  $h \leftarrow hash(\mathsf{head}_\ell)$
  $w \leftarrow \{0,1\}^\kappa$
  $\mathsf{head}_{\ell+1} := \varepsilon$
  if $hash(h, \tau, w) < \mathrm{T}$ **then**
    $\mathsf{head}_{\ell+1} := \langle h, \tau, w \rangle$
  end if
  return $\mathsf{head}_{\ell+1}$
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 $Validate(\ \mathsf{C}\ )$
  $\langle \mathcal{H}_\ell, \bar{\mathbf{x}}_\mathsf{C}, \tilde{x}_\mathsf{C} \rangle \leftarrow \mathsf{C}$
  $b := \mathsf{V}(\bar{\mathbf{x}}_\mathsf{C})$
  if b **then**
    $\langle \mathsf{head}_1, \mathsf{head}_2, \ldots, \mathsf{head}_\ell \rangle \leftarrow \mathcal{H}_\ell$ ▷ the length of C is ℓ
    $h' := 0$
    for each i in $1, \ldots, \ell$ **do**
      parse head_i into $\langle h_i, \tau_i, w_i \rangle$
      if $(h_i = h') \wedge (hash(\mathsf{head}_i) < \mathrm{T})$ **then**
        $h' \leftarrow hash(\mathsf{head}_i)$
      else
        return $(b = 0)$
      end if
    end for
  end if
  return (b)
end function

Algorithm 5 Best chain function $BestChain$.

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

B.2 Security analysis for the blockchain

The underlying header-chain in the blockchain is the same as the Bitcoin blockchain in [7] 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 [7], the blockchain protocol $\Pi_{\mathcal{C}}$ (Section B.1) achieves the three security properties: *common-prefix property*, *chain quality property*, and *chain growth property*.

We begin by recalling the following two quantities introduced in [7, 17]. Consider the total number of players is n, the portion of malicious computing power is ρ, and $p = \frac{\tau}{2^\kappa}$ 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 [17]. 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 B.1 (Chain growth). *For any $\delta, \gamma > 0$, consider the blockchain protocol $\Pi_{\mathcal{C}}$ (see Section B.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 $\text{VIEW}_{\Pi_{\mathcal{C}},\mathcal{A},\mathcal{Z}}$, the probability that $\ell' - \ell \geq g \cdot s$ is at least $1 - e^{-\Omega(s)}$ where $g = (1-\delta)\gamma$.*

Theorem B.2 (Chain quality). *Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. For any $\delta > 0$, consider the blockchain protocol $\Pi_{\mathcal{C}}$ (see Section B.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 $\text{VIEW}_{\Pi_{\mathcal{C}},\mathcal{A},\mathcal{Z}}$, 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 B.3 (Common prefix). *Assume that $\gamma = \lambda\beta$ and $\lambda > 1$. For any $\delta > 0$, consider the blockchain protocol $\Pi_{\mathcal{C}}$ (see Section B.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 $\text{VIEW}_{\Pi_{\mathcal{C}},\mathcal{A},\mathcal{Z}}$ 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 $\Pi_{\mathcal{C}}$ satisfy the chain growth, chain quality, and common-prefix properties.

C Supporting material for Section 5

C.1 Building block: Authenticated data structure

To authenticate a block via a compact header Bitcoin is using Merkle tree [12], which 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 transaction set.

Definition C.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 C.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*

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

where $\mathtt{Auth\text{-}coll}_{\mathcal{A},\Sigma_{\mathsf{au}}}(\kappa)$ 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.*

C.2 Proof of Theorem 5.2

Proof. By Theorem B.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 C.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. We will construct $\mathcal{A}'$ from the execution of the full-mode Π^{full} that is directed by $\mathcal{Z}$ as follows. The adversary $\mathcal{A}'$ is given 1^κ. He then randomly chooses pair of honest players P_1 with the snapshot $\mathtt{SS}^1_{\ell_1} := \mathcal{T}^1_{\ell_1}$ of length ℓ_1 and P_2 with the snapshot $\mathtt{SS}^2_{\ell_2} := \mathcal{T}^2_{\ell_2}$ of length $\ell_2 = \ell_1$, 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 choose random $\ell \in [\ell_1]$, and 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 $\Pr[\mathcal{T}^i_{\ell_i} \neq \mathcal{T}^j_{\ell_j}] > \zeta$.

Thus,

$$\Pr[\mathcal{T}^2_{\ell_2} \neq \mathcal{T}^1_{\ell_1}] > \frac{\zeta}{n^2} \text{ and } \Pr[x^2_\ell \neq x^1_\ell] > \frac{\zeta}{n^2 q}$$

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 C.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_ℓ^1, x_ℓ^2) such that $x_\ell^1 \neq x_\ell^2$, $\mathsf{Root}(x_\ell^1) = \mathsf{Root}(x_\ell^2) = \tau_\ell^1 = \tau_\ell^2$, $\mathsf{CheckRoot}(x_\ell^1, \tau_\ell^1) = \mathsf{CheckRoot}(x_\ell^2, \tau_\ell^2) = 1$, with probability at least ζ' where $\zeta' = \frac{\zeta}{n^2 q}$. This completes the proof. □

C.3 Proof of Theorem 5.3

Proof. By Theorems B.2 and B.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 $\mathtt{tx}$ for at most $t = (1+\delta)\frac{2\kappa}{\gamma}$ rounds, for $\delta > 0$, then for any player P there exists snapshots $\mathtt{SS}_\ell := \mathcal{T}_\ell$ of length ℓ, $\mathtt{SS}_{\ell-1} := \mathcal{T}_{\ell-1}$ of length ℓ-1 and a transaction set x, in the full mode, such that $\mathtt{SS}_\ell := \mathtt{SS}_{\ell-1} \diamond x$ and $\mathtt{tx} \in x$, with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function and $\diamond := ||$.

Note that, by the chain-growth property (Theorem B.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 B.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 $\Sigma_{\mathtt{au}}$ that can win the collision-resistant experiment (see Definition C.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.

We then construct the adversary $\mathcal{A}'$ from the execution in the full-mode that is directed by $\mathcal{Z}$ 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 $\mathtt{SS}_\ell := \mathcal{T}_\ell$ 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 $\mathtt{SS}_\ell := \mathcal{T}_{\ell-1}$ 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 $\mathtt{SS}_\ell := \mathtt{SS}_{\ell-1} \diamond x'$, 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'. This implies that 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 $\Sigma_{\mathtt{au}}$ can win the collision-resistant experiment (see Definition C.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. □

D Supporting material for Section 6

D.1 Building block: UTXO-set

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, 13] or spent transaction outputs[4]. For a payment to be 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 $\mathcal{U}$. Now, instead of storing all historical and unnecessary transactions, miners are expected to store a UTXO-set.

Next, we define a *sequence of UTXO-sets*, which will help us present our construction. Recall that, without loss of generality, we treat an output of a transaction just as a unique bitstring in $\{0,1\}^*$. Then $\mathcal{U}$ is a set of outputs not yet spent. An application of transaction $\mathtt{tx}$ to $\mathcal{U}$ removes outputs spent by $\mathtt{tx}$ from $\mathcal{U}$, and the application fails if any of the outputs spent by the transaction is not found in the UTXO-set $\mathcal{U}$, and then adds newly created by $\mathtt{tx}$ outputs to $\mathcal{U}$. This idea can straightforwardly extended for sorted transaction set x, and pseudo-code for this procedure can be found in the Algorithm 6.

Algorithm 6 UTXO operation $\circ$.

```
1:  function ∘ (U_ℓ, x_{ℓ+1})
2:      U_{ℓ+1} := U_ℓ
3:      for each transaction tx in x_{ℓ+1} do          ▷ transactions are stored in temporal order
4:          for each transaction input Inp in tx do
5:              if Inp ∉ U_{ℓ+1} then
6:                  return ⊥
7:              else
8:                  remove Inp from U_{ℓ+1}
9:              end if
10:         end for
11:         add all transaction outputs in tx to U_{ℓ+1}
12:     end for
13:     return U_{ℓ+1}
14: end function
```

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 6. Similarly, the first UTXO-set $\mathcal{U}_1$ can be further updated into the secondUTXO-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 $x_{\ell+1}$. Note that, 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 $\mathcal{U}^j$. Formally, we have

Definition D.1 (UTXO-set sequence)**.** *A sequence of UTXO-sets $\mathbb{U} = (\mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_\ell)$, for $\ell \in \mathbb{N}$, is defined by an ordered sequence of transaction sets $x_1, x_2, \ldots, x_\ell$, and initial state $\mathcal{U}_0$, together with an operation $\circ$, as follows: (1) $\mathcal{U}_0$ is a publicly known constant, and (2) $\mathcal{U}_i := \mathcal{U}_{i-1} \circ x_i$ for all $i \in [\ell]$.*

D.2 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 $\Sigma_{\mathtt{au}}$ (see Definition C.1) is collision-resistant, and that $\gamma = \lambda\beta$ and $\lambda > 1$.

By Theorem B.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.

[4] An output is *spent* if it is referenced by some transaction in the history, otherwise it is *unspent*.

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 $1 - \epsilon(\kappa)$

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

□

D.3 Proof of Theorem 6.2

Proof. We prove by induction as follows. Initially, we have $\mathtt{SS}^1_\emptyset = \mathtt{SS}^2_\emptyset = \mathcal{U}_0$. 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. This completes the proof.

□

Theorem D.2 (prune-mode liveness)**.** *Consider the generalized ledger protocol Π^{prune} (see Section 6.1). Let κ be the security parameter. Assume the authenticated data structure $\Sigma_{\mathtt{au}}$ (see Definition C.1) is collision-resistant. 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. The proof for this theorem is straightforward. □

E Supporting material for Section 7

E.1 Proof of Lemma 7.1

Proof. Let $\mathcal{S}_{\ell-\kappa}$ be the set of all spent transaction outputs in $\mathcal{T}_\ell$, and let $\mathcal{G}_{\ell-\kappa}$ be the set of all transaction outputs in $\mathcal{T}_\ell$, i.e., $\mathcal{G}_{\ell-\kappa} := \mathcal{S}_{\ell-\kappa} \cup \mathcal{U}_{\ell-\kappa}$. Let $\mathtt{SS}_\ell^{\text{temp}} := \mathcal{G}_{\ell-\kappa}$. We first prove that

$$\mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\ell^{\text{temp}} \diamond^{\mathsf{p}} x) = \mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\ell^{\mathsf{p}} \diamond^{\mathsf{p}} x) \tag{1}$$

where $\mathtt{SS}_\ell^{\mathsf{p}} := \mathcal{U}_{\ell-\kappa}$ and $\diamond^{\mathsf{p}} := \circ$. Now, there are two cases:

- *Case 1:* If the transaction inputs of a valid transaction $\mathtt{tx}$ in x are in $\mathcal{U}_{\ell-\kappa}$. Then Equation (1) is true since $\mathcal{U}_{\ell-\kappa} \subseteq \mathcal{G}_{\ell-\kappa}$.
- *Case 2:* If the transaction inputs of a valid transaction $\mathtt{tx}$ in x are not in $\mathcal{U}_{\ell-\kappa}$. In any cases where the transaction inputs are in $\mathcal{S}_{\ell-\kappa}$ or not, since $\mathcal{S}_{\ell-\kappa}$ is the set of all spent transaction outputs, the operation "$\circ$" will output $\perp$, then data validation predicate in prune mode (See Table 2) will return false.

From the two cases, it follows that Equation (1) is true. Then, by the definition of the predicate $\mathbb{V}_{\text{FULL}}$ (See Table 1), it follows that

$$\mathbb{V}_{\text{FULL}}(\mathtt{SS}_\ell^{\mathsf{f}} \diamond^{\mathsf{f}} x) = \mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\ell^{\text{temp}} \diamond^{\mathsf{p}} x) \tag{2}$$

where $\mathtt{SS}_\ell^{\mathsf{f}} := \mathcal{T}_\ell$ in full and $\diamond^{\mathsf{f}} := ||$. From Equations (1) and (2), we conclude that, for any transaction set x,

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

□

E.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 $\mathtt{SS}_\ell^{\mathsf{f},1} := \mathcal{T}_\ell^1$ and the snapshot of P_2 in full mode $\mathtt{SS}_\ell^{\mathsf{f},2} := \mathcal{T}_\ell^2$. By Theorem 5.2, 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

  $$\mathbb{V}_{\text{FULL}}(\mathtt{SS}_\ell^{\mathsf{f},1} \diamond^{\mathsf{f}} x) = \mathbb{V}_{\text{FULL}}(\mathtt{SS}_\ell^{\mathsf{f},2} \diamond^{\mathsf{f}} x)$$

  with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function and $\diamond^{\mathsf{f}} := ||$.
- *Case 2*: If both P_1 and P_2 are in prune mode. Here, the snapshot of P_1 in prune mode $\mathtt{SS}_\ell^{\mathsf{p},1} := \mathcal{U}_{\ell-\kappa}^1$ and the snapshot of P_2 in prune mode $\mathtt{SS}_\ell^{\mathsf{p},2} := \mathcal{U}_{\ell-\kappa}^2$. 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

  $$\mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\ell^{\mathsf{p},1} \diamond^{\mathsf{p}} x) = \mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\ell^{\mathsf{p},2} \diamond^{\mathsf{p}} x)$$

  with probability at least $1 - \epsilon(\kappa)$, where $\epsilon(\cdot)$ is a negligible function and $\diamond^{\mathsf{p}} := \circ$.
- *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 $\mathtt{SS}_\emptyset^{\mathsf{f},1} = \epsilon$ and $\mathtt{SS}_\emptyset^{\mathsf{p},1} = \mathcal{U}_0$. By definitions of the predicates $\mathbb{V}_{\text{FULL}}(\cdot)$ and $\mathbb{V}_{\text{PRUNE}}(\cdot')$, it holds that $\mathbb{V}_{\text{FULL}}(\mathtt{SS}_\emptyset^{\mathsf{f},1}) = \mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\emptyset^{\mathsf{p},2}) = 1$, where $\mathtt{SS}_\emptyset^{\mathsf{f},1} = \epsilon$ and $\mathtt{SS}_\emptyset^{\mathsf{p}} = \mathcal{U}_0$ denote the initial snapshots in prune and full modes, respectively.

  By the full mode snapshot persistence (Theorem 5.2), 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 $\mathtt{SS}_\ell^{\mathsf{f},1} := \mathcal{T}_\ell^1 =$

$\langle x_1^1, \ldots, x_{\ell-\kappa}^1 \rangle$, and the snapshot of P_2 in prune mode $\mathtt{SS}_\ell^{\mathsf{p},2} := \mathcal{U}_{\ell-\kappa}^2 = \mathcal{U}_0 \circ x_1^2 \circ \ldots \circ x_{\ell-\kappa}^2$. From Lemma 7.1, we conclude that, for any transaction set x, with probability at least $1 - \epsilon(\kappa)$,

$$\mathbb{V}_{\text{FULL}}(\mathtt{SS}_\ell^{\mathsf{f},1} \diamond^{\mathsf{f}} x) = \mathbb{V}_{\text{PRUNE}}(\mathtt{SS}_\ell^{\mathsf{p},2} \diamond^{\mathsf{p}} x)$$

This completes the proof.

□