securePrune: Secure block pruning in UTXO based blockchains using Accumulators

Swaroopa Reddy B
Department of Electrical Engineering
Indian Institute of Technology Hyderabad
Hyderabad, India
ee17resch11004@iith.ac.in

Abstract—In this paper, we propose a scheme called securePrune for reducing the storage space of a full node and synchronization time of bootstrapping nodes joining the Peer-to-Peer (P2P) network in an Unspent Transaction Outputs (UTXO) based blockchain like bitcoin using RSA accumulators. The size of the bitcoin blockchain is growing linearly with transactions. We propose a new block structure to represent the state of a blockchain also called UTXO set by including an accumulator of a state in the block header and proofs of knowledge for inclusion and deletion of the transactions of the current block in the block. In our scheme, the miners periodically release a snapshot of the blockchain state. The other full nodes in the network, securely prune the historical blocks after attaining the required number of confirmations for the snapshot block, which in turn confirms the snapshot of the state through an accumulator specified in the block header and proofs inside the block. The secure and periodic pruning of the old blocks, reduce the synchronization time for a new node joining into the network. The simulation results demonstrate a significant reduction in the storage space of a full node and bootstrapping cost of the new nodes.

Index Terms—Blockchain, UTXO, RSA Accumulator, Pruning, Inclusive Proofs, Bootstrapping.

I. INTRODUCTION

The Blockchain is a revolutionary technology behind the Peer-to-Peer (P2P) cryptocurrency networks like bitcoin [1] and smart contract enabled P2P networks like Ethereum [2] and Hyperledger [3]. The decentralized and trustless nature of the blockchain created a space for applications in healthcare [4], Internet of Things (IoT) [5].

The transactions are the fundamental entities in the blockchain which represents the transfer of coins from one party to another. The miner collects the multiple transactions from other miners/fullnodes and creates a block with computationally hard problem called Proof-of-Work (PoW) [1]. The immutability of the blockchain ledger lies in PoW, as any modification to the ledger by a miner needs to recompute all the blocks till the current position of the chain. To achieve the consensus among all the nodes, while creating a new node the miner selects a tip of the longest chain in the network.

The bitcoin blockchain is a P2P network of miners [1], fullnodes [9] and simplified payment verification nodes (SPV) [1]. The miners play a key role in generating the blocks through PoW puzzle. The full node stores all the blocks since the genesis block along with the blockchain state (UTXO set) [7]. The UTXO set keep track of the all unspent output transactions of the historical blocks and used as sources for new input transactions. A full node contributes to the security of the network through block validation. However, running a full node incurs storage costs as the blockchain data grows exponentially [9] with time. The main advantage of storing the all blocks by the full node is to make the bootstrapping nodes synchronize with the existing network nodes.

Erasure-code based low storage blockchain node is proposed in [12]. These nodes split every block into fixed size datafragments and generates the coded fragments from the linear combination of the random coefficients generated from the pseudo-random generator. The node can reconstruct the block by the inverse linear combinations. The main limitation of this work is that they only consider the case when nodes can leave the network or can be unreachable, they do not consider adversarial nodes that can provide maliciously formed coded fragments.

A Secure Fountain architecture founded on coding theory is proposed in [13] for storage efficiency of the blockchain node by reducing the storage cost and still contribute to bootstrapping a new node joining the network. In this scheme, the nodes reduce the storage cost by encoding the validated blocks into a small number of coded blocks using fountain codes [14]. The secure Fountain (SeF) architecture uses header-chain as a side information to check whether the decoded blocks are formed from the malicious modifications.

In [15], authors proposed a Dynamic distributed storage for scaling the blockchains by allocating the nodes into dynamic zones. The nodes in each zone will store a share of private key using shamir's secret sharing [16] for encrypting the block data and apply a distributed storage codes such as [17], [18] for reducing the storage cost.

However, In all these works, a bootstrapping node need to download all the blocks in the form of distributed coded fragments and validate all the decoded blocks to synchronize with the existing nodes.

In this paper, we propose a periodic pruning of the historical blocks based on the security confirmations guaranteed by the RSA accumulator [10], [11] of the UTXO set and PoW based longest chain consensus algorithm [1].

This research was funded by Indigenous 5G Test Bed (Building an end to end 5G Test Bed) in India, Department of Telecommunication Network & Technologies (NT) Cell.
The main contributions of the paper are as follows.

1) The algorithm for block generation by a miner by adding accumulator for state (UTXO set) in the block header to make the state as a part of the PoW consensus algorithm. The miner also includes Non-Interactive proof of Exponentiation (NI-PoE) proofs for inclusion of the new output transactions and deletion of the UTXO sources of the input transactions of the current block.

2) The algorithm for validation of a block by every full node based on the NI-PoE proofs added by the miner inside the block.

3) The periodic pruning of the blocks at regular intervals of the block height based on the security guaranteed by the accumulator state and the NI-PoE proofs.

4) The bootstrapping procedure for synchronizing the new nodes joining the network.

Through the event-driven simulation of blockchain, we have shown the 85% reduction in the storage space of a securePrune protocol full node compared to the bitcoin full node and also significant reduction in the synchronization time due to the requirement of validation of less number of historical blocks compared to the validation of all the historical blocks in the bitcoin.

The rest of the paper is organized as follows. In Section II, we present the related work in the reduction of the storage space of a bitcoin full node. Section III gives the system model and notations used in the protocol. Section IV describes the preliminaries for generation of accumulator state and NI-PoE proofs. In Section V, we describe the proposed protocol for secure and periodic pruning and synchronization of the bootstrapping nodes. In Section VI, we present and discuss the simulation results. Section VII presents the concluding remarks and future works.

II. RELATED WORK

The SPV node or light weight node keep the block headers of a longest PoW chain, instead of the entire blockchain data. The lightweight node depends on the full node for the verification of a transaction by querying the merkle branch linking the transaction to block where it is time stamped. The block pruning is allowed in bitcoin to store most recent blocks of the chain. However, due to lack of old blocks like in full node, pruning node cannot serve the new full nodes joining the network. The concept of assumed-valid blocks have been introduced in bitcoin, where a bootstrapping node skips the script validation of the transactions for ancestors of known-good blocks, without changing the security model. However, the new nodes still need to download entire historical data to create the current state of the blockchain.

In coinPrune protocol, the authors proposed the pruning older blocks by creating a snapshot of the state at regular intervals, provided the collective reaffirmations to snapshot by the miners. In this protocol, the fullnode prunes the historical blocks provided that the snapshot receives the required number of reaffirmations from all the miners. However, there is a possibility of the Denial-of-Service (DOS) attack by the miners in reaffirming the snapshot. So, there is no guarantee that pruning will happen at every reaffirmation window of a snapshot release.

III. SYSTEM MODEL AND PARAMETERS

The parameters used in securePrune protocol are listed in TABLE I.

### TABLE I:

| Symbols | Description |
|---------|-------------|
| Hash( ) | Cryptographic hash function |
| root( ) | Merkle root of set of transactions |
| H_{prime}( ) | Prime representative function |
| validate( ) | Transaction validation function |
| R_i | Merkle root of t_i |
| B_i | i^{th} block of the blockchain |
| M | Memory pool/set of unconfirmed transactions |
| tx | Transaction |
| t_i | (tx_1, tx_2, ... ,tx_{t_i}); Set of transactions in B_i |
| S_i | State of the block chain (UTXO set) at block B_i |
| S_d | Set of utxo’s spent in the new block |
| S_a | Set of output transactions in the new block B_i |
| A_i | Accumulator state |
| W | List of membership witnesses of UTXO set |
| π_d | NI-PoE proof for deletion of set S_d from accumulator |
| π_a | NI-PoE proof for addition of set S_a to accumulator |
| T_{prune/f} | NI-PoE verification time |
| Δ_s | Number of blocks between two snapshots |
| n | Number of nodes in the network |
| q | Fraction of the attacker’s hashrate |
| r_p | Number of peers connected to each node |
| T_p | Propagation delay |
| b | Block size in MB |
| R | Average download bandwidth |
| R_v | Average validation rate of a block |
| p_v | Computational power with node v |
| λ | Block creation rate |
| D | End-to-end delay in the network |
| k | Number of confirmations |
| h | Height of the blockchain |
| m | Number of mining nodes in the network |

A. Overview of the transactions and UTXO set

There are two types of the transactions in every transaction of a block - inputs and outputs. The inputs specifies the previous transaction outputs as sources of the bitcoins in a transaction and the outputs are the destination of bitcoin transfer. Each transaction contains multiple inputs and multiple outputs to combine and split the values of the coin transfer. The full node stores the UTXO set in the chainstate database of the Bitcoin core. The database consists of records of key-value pairs. The key of the record is transaction hash and the value stores the transaction information. Every record
in the UTXO set represent the outputs yet to be spent in future transactions.

Let at a block height $i$, every full node in the blockchain stores a copy of the state (UTXO set) $S_i$ represented as

$$S_i = \{u_j : j = 1, 2, \ldots, |S_i|\}$$

(1)

Where, $u_j$ is a record in the UTXO set

However, for every new block ($B_i$) addition to the chain, the state of the full node changes with the transaction set $t_i$ as described in state transition Algorithm in Section V-A1

B. The modified block structure in the proposed protocol

The blockchain at a height $h$ is modeled as a vector of blocks represented as

$$C_h = (B_0, B_1, \ldots, B_h)$$

(2)

where, each block $B_i$ is a tuple consists of block header ($H_i$), NI-PoE proof ($\pi_d$) for deletion of set $S_d$ from the UTXO set and NI-PoE proof ($\pi_a$) for addition of set $S_a$ to the UTXO set. While generating a new block, the miner includes a list of transactions $t_i$ into the block from the transaction memory pool ($\mathcal{M}$) stored with every miner and full nodes.

$$B_i = < H_i, (A'_{i-1}, \pi_d, \pi_a), t_i >$$

(3)

Where, the tuple $(A'_{i-1}, \pi_d, \pi_a)$ results from the state transition of the UTXO set.

In addition to the elements of the bitcoin block header, our proposed model includes an extra element called the accumulator state ($A_i$), which is an RSA accumulator [10] to represent the state of the blockchain ($S_i$) in the block header.

$$A_i = (h_{i-1}, nonce, A_i, x)$$

(4)

where, $h_{i-1} = hash(H_{i-1})$, nonce is a variable to solve the PoW puzzle and $x$ is the other meta data (like version, time, difficulty etc) similar to bitcoin block header [1]. The modified structure of the block is shown in Fig. 1.

IV. PRELIMINARIES

The following definitions of RSA accumulators [10], [11] are used in our work.

Definition 1. (Accumulator of State). Let $\mathbb{G}$ be a group of unknown order and $g \in \mathbb{G}$, the accumulator state of a block $B_i$ is an RSA accumulator [10] of the unspent transaction outputs present in UTXO set $S_i = \{u_j : j = 1, 2, \ldots, |S_i|\}$ and is computed as

$$A_i = \prod_{j=1}^{S_i} g^{U_j}$$

(5)

Where, $U_j$ is the prime representative of the element $u_j$ [11]

$$U_j = H_{\text{prime}}(u_j)$$

(6)

The dynamic accumulator [21] is an accumulator that allows to add or delete elements to the accumulator.

Definition 2. (Membership Witness). A membership witness is simply an accumulator without an aggregated element. The membership witness for $u_m$ is defined as

$$W_m = \prod_{j=1, j \neq m}^{S_i} g^{U_j}$$

(7)

Definition 3. (Shamir Trick). Let $x, y \in S$ and $g \in \mathbb{G}$, the membership witnesses for $x$ and $y$ are the $x^{th}$ root of $g$ and $y^{th}$ root of $g$, then the Shamir Trick [22], [11] is a $(xy)^{th}$ root of the group element $g$ from the Bezout's coefficients of $x$ and $y$.

While creating a new block, the miner generates new accumulator state $A_i$ from $A_{i-1}$ in two stages - The deletion of the set $S_d$ from the accumulator $A_{i-1}$ followed by addition of set $S_a$ to obtain the new state $A_i$.

$$A_i = \text{BatchAdd}(\text{BatchDel}(A_{i-1}, S_d), S_a)$$

(8)

Definition 4. (Batch Deletion (BatchDel)). Let $S_d$ represent the set of sources for inputs of the transactions in the new block $B_i$, the state $S_{i-1}$ needs to delete the records $S_d$ from the database. The deletion of the set $S_d$ from accumulator state $A_{i-1}$ can be obtained from BatchDel [11]. The BatchDel uses the membership aggregation function $\text{AggMemWit}$ [11] to compute the aggregate membership witness of all elements in $S_d$ from the individual membership witnesses of each element. The AggMemWit is simply an accumulator without elements of set $S_d$.

$$A'_{i-1} = \text{W_{agg}} = \prod_{u_j \in S_{i-1} \backslash S_d} g^{U_j}$$

(9)
where, $W_{agg}$ is the aggregated membership witness of all the elements of the set $S_d$ generated by Shamir Trick 11. The $BatchDel$ gives the intermediate state of the accumulator $A_{i-1}'$ to process further for obtaining the new accumulator state ($A_i$) of the new block $B_i$ from the set $S_i$.

**Definition 5. (Batch Addition (BatchAdd)).** The addition of the elements of set $S_i$ to accumulator state requires a batch addition $BatchAdd$ 11 for efficient computation.

$$A_i = (A_{i-1}')^{U^*}$$  

where,

$$U^* = \prod_{s \in S_i} H_{\text{prime}}(s)$$

**Definition 6. (Proof of Exponentiation (PoE) 11).** Let $\mathbb{G}$ be a group of unknown order and $u, w \in \mathbb{G}$, the proof of exponentiation in the Group $\mathbb{G}$, when both the prover and verifier are given $(u, w, x \in \mathbb{Z})$ and the prover wants to convince the verifier that $w = u^x$.

The Non-interactive PoE (NI-PoE 11) proofs $\pi_d$ and $\pi_a$ are generated during the batch updates for the efficient verification without any interaction between prover(miner) and verifier(full node).

$$\pi_d = NI - PoE(u, x, w)$$

**Definition 7. (Updating membership witnesses).** The intermediate accumulator state 12 denotes the membership witness for all the elements of the set $S_d$. Let $s \in S_{i-1} \setminus S_d$ and $w_s$ is the membership witness of $s$ before deletion of set $S_d$ as per (7), then the updated membership witnesses for all $s \in S_{i-1} \setminus S_d$ are generated as follows

$$w'_s = ShamirTrick(A_{i-1}', w_s, \prod_{x \in S_d} x, s)$$

The membership witness updates for all $s \in S_{i-1} \setminus S_d$ after the addition of elements of the set $S_a$ are calculated as follows

$$w''_s = (w'_s)^{\prod_{y \in S_a} x}$$

The membership witnesses for elements $x \in S_a$ are calculated as follows

$$w_x = (A_{i-1}'')^{\prod_{y \in S_a, x \neq y}}$$

**V. SECURE BLOCK PRUNING PROTOCOL**

In this section, we discuss the proposed secure pruning protocol for storage scalability and the synchronization process of the bootstrapping nodes. The protocol requires the modification in the block generation procedure by the miners and the validation procedure of a block by the full nodes in the network based on the accumulators and NI-PoE.

**A. Requirements of the securePrune protocol**

1) **State transition Algorithm**: The UTXO set (state) of the blockchain is dynamic and changed for every new block addition to the blockchain. The following algorithm describes the transition of a miner (or full node) while generating a new block or after receiving a new block. The new state transition function returns the set of deleted elements ($S_d$) and added elements ($S_a$) along with the new UTXO set.

**Algorithm 1 State transition Algorithm**

**Input:** $S_{i-1}, t_i$

**Output:** $S' -$ new state, $S_d, S_a$

1: **procedure** STATETRANSITION ($S_{i-1}, t_i$)
2: $S' \leftarrow S_{i-1}$
3: for $tx$ in $t_i$ do
4: \hspace{0.5cm} isValid $\leftarrow$ validate($tx$)
5: \hspace{0.5cm} if $isValid$ then
6: \hspace{1.0cm} for input in $tx$ do
7: \hspace{1.5cm} id $\leftarrow$ input[$txHash$]
8: \hspace{1.5cm} delete $u_j[id]$ from $S'$
9: \hspace{1.5cm} $S_d$.append($u_j[id]$)
10: \hspace{1.0cm} end for
11: \hspace{1.0cm} for output in $tx$ do
12: \hspace{1.5cm} $S'$.append($output$)
13: \hspace{1.5cm} $S_a$.append($output$)
14: \hspace{1.0cm} end for
15: \hspace{0.5cm} else
16: \hspace{1.0cm} return False
17: \hspace{0.5cm} end if
18: \hspace{0.5cm} end for
19: return $S'$, $S_d$, $S_a$
20: **end procedure**

2) **Modified PoW Algorithm**: The modified Proof-of-Work function for mining a new block is described in Algorithm 2. This PoW function includes Accumulator state $A_i$ along with other parameters into the block header for providing immutable blockchain state $Si$. It also includes NI-PoE proofs ($\pi_d$, $\pi_a$) for deletion and addition of the new set of elements ($S_d$, $S_a$) to the state from the present transaction set $t_i$.

The NI-PoE proof $\pi_d$ is obtained from the $BatchDel$ function 11 as a proof for deletion of the unspent transactions referred in the inputs of the set $t_i$. The $BatchDel$ function deletes the sources of inputs ($S_d$) from the accumulator state of the previous block $A_{i-1}$ and generates the NI-PoE proof $\pi_d$. The proof $\pi_a$ is also an NI-PoE proof generated from the $BatchAdd$ function 11 for adding the outputs ($S_a$) of set $t_i$.

3) **Block Validation Algorithm**: We defined a validation function in Algorithm 3 to check the validity of $A_i$, $t_i$, $R_i$, $\pi_d$ and $\pi_a$ from the present state $S_i$, local chain $C_{i-1}$ and the received new block ($B_i$). If $B_i$ is valid, the full node adds $B_i$ to $C_{i-1}$, otherwise discards the block.

**B. securePrune Protocol**

The protocol differs from the bitcoin protocol by issuing a snapshot of the UTXO set at regular intervals of every $\Delta_s$ blocks called snapshot interval. The miners while creating a new block as per the Algorithm 2 at a height $c\Delta_s$ ($c = 1,2,3,\ldots$) releases the snapshot along with the block $B_p$ created at that particular height. The snapshot consists of an identifier and a copy of the state ($S_p$) consists of all the unspent transactions including the unspent transactions of the current block. The snapshot identifier is the accumulator state
Algorithm 2 The modified PoW function for the secure prune protocol

Input: $S_{i-1}$, $C_{i-1}$, $M,W$
output: $C_i$

1: procedure securePrunePoW($S_{i-1}$, $C_{i-1}$)
2: for $tx$ in $M$ do
3: $t_i$,append($tx$)
4: if size of $B_i$ > Max Block Size then break
5: end if
6: end for
7: $S_i, S_d, S_a$ ← stateTransaction($S_{i-1}, t_i$)
8: $A'_{i-1}, \pi_d$ ← BatchDel($A_{i-1}, S_d, W$)
9: $A_i, \pi_a$ ← BatchAdd($A'_{i-1}, S_a$)
10: $W' = updateMemWit(A'_{i-1}, W, S_d, S_a)$
11: $nonce ← 0$
12: while $nonce < 2^{32}$ do
13: $h ← hash(H_{i-1}, R_{t_i}, A_i, x)$
14: if $hash(nonce, h) > Difficulty$ then break
15: end if
16: $nonce ← nonce + 1$
17: end while
18: $W$ ← $W'$
19: $H_i ← < H_{i-1}, R_{t_i}, A_i, x, nonce >$
20: $B_i ← < H_i, \pi_d, \pi_a, t_i >$
21: $C_i ← C_{i-1} B_i$
22: return $C_i$
23: end procedure

Algorithm 3 Block Validation Algorithm

Input: $S_{i-1}$, $C_{i-1}$, $B_i$
output: $C_i, S_i$

1: procedure validateBlock($S_{i-1}, C_{i-1}, B_i$)
2: $t_i$ ← $B_i[t_i]$
3: $count ← 0$
4: for $tx$ in $t_i$ do
5: $isValid ← validate(tx)$
6: if not $isValid$ then return False
7: end if
8: $count ← count + 1$
9: end for
10: if $R_i ≠ root(t_i)$ then return False
11: end if
12: if count == $|t_i|$ then
13: $A'_{i-1}, \pi_d, \pi_a$ ← $B_i$
14: $A_{i-1}$ ← $B_i [accState]$
15: $S_i, S_d, S_a$ ← stateTransaction($S_{i-1}, t_i$)
16: $a$ ← $NI–PoE. Verify(\prod_{s \in S_a} s, A'_{i-1}, A_{i-1}, \pi_d)$
17: $b$ ← $NI – PoE. Verify(\prod_{s \in S_d} s, A'_{i-1}, A_i, \pi_a)$
18: if $a \wedge b$ then $S_i$ ← $S'$
19: $C_i ← C_{i-1} B_i$
20: end if
21: return $C_i$
22: end procedure

present in the block header of snapshot block $B_p$. The chain subsequent to the snapshot block $B_p$ is termed as the tailchain. The full node follows Algorithm 3 for validation of a block created during $\Delta_s$ (present in the tailchain) by verifying the NI-PoE proofs $\pi_d$ and $\pi_a$, merkle root $R_i$ and transactions $t_i$.

The full nodes in the network prune all the historical blocks prior to the snapshot block $B_p$ provided that the block $B_p$ achieved $k$ number of confirmations from the tailchain blocks created in the network. The full nodes choose the tip of the longest chain similar to bitcoin for deciding the number of confirmations on $B_p$. Suppose, more than one miner releases a snapshot at a height $p + c\Delta_s$, the snapshot with longest tailchain is a valid snapshot. Fig. 2 describes the overview of the securePrune protocol.

**Lemma 5.1:** Let $k$ be the number of confirmations required for a block with very low probability of double-spend to succeed by an attacker, then the number of confirmations required for a snapshot is also $k$.

**Proof:** If any attacker tries to modify a transaction in a block, the hash of the block change as the merkle root is a function of all the transactions in a block. The doublespend attack is the creation of a secret chain longer than the chain with the original transaction. So, an attacker needs to create a chain longer than the honest chain to modify any transaction of a particular block. The probability of double-spend to be succeed by an attacker with a fraction of hashrate $q$ for a given number of confirmations is shown in [11] and [23].

Suppose, a miner in the network creates a block at height $p$ and has received $k$ number of confirmations. Let $B_{p+1}, B_{p+2}, \ldots, B_{p+k}$ are the blocks that confirms block $B_p$. A transaction in the block is said to be a part of the valid chain, if it has $k$ number of confirmations with very low probability of double-spend by an attacker.

Let $\bullet$ denotes the state transition function described in Algorithm 1, the new state of a snapshot $S_p$ after $k$ number of blocks is appended to block $B_p$ is given by
Synchronization process depends on average download rate \((R)\) and average block validation Rate \((R_v)\) since the state is represented as a part of the PoW function in Algorithm 2 in terms of the accumulator state \(A_i\) for a block \(B_i\), the hash of the block changes with change in the state particularly snapshot.

Thus, the number of confirmations required for immutability of the snapshot is same as the number of confirmations required for a block against the double-spend attack. ■

C. Size of the Blockchain in securePrune

Fig. 3 shows the total blocks from block \(B_p\) to the current height including \(B_{p+\Delta_s}\), block at snapshot \(S_{p+\Delta_s}\). Suppose, the nodes pruned the blocks till height \(B_{p-1}\) after achieving the required number of confirmations to snapshot \(S_p\), then every node in the network stores the blocks from \(B_p\) onwards till the current height.

Suppose, a miner broadcast a block into the network at \(p + \Delta_s\) along with snapshot \(S_{p+\Delta_s}\), then the total number of blocks till block height \(p + \Delta_s + k - 1\) are \(\Delta_s + k\). At height \(p + \Delta_s + k\), the nodes prune the blocks \(B_p\) to \(B_{p+\Delta_s-1}\). So, the total number of blocks stored with a node is upper bounded by \(\Delta_s + k\).

D. Synchronization of the Bootstrapping nodes

The new node joining the network bootstrap in three steps - First, it obtains the most recent snapshot \(S_p\) and a node joins the network at height \(h\), then the state of the new node at height \(h\) is obtained as

\[
S_h = S_p \cdot B_{p+1} \cdot \cdots \cdot B_h
\]

Let \(b, c, s\) are the new nodes joining in bitcoin, coinPrune and securePrune respectively, then, \(n_b, n_c, n_s\) are the number of blocks to be downloaded by nodes \(b, c\) and \(s\). The synchronization process depends on average download rate \((R)\) and block size \(b\) (assuming a constant block size). The synchronization time required for these new nodes joining these networks are defined as follows

\[
T_b = n_b \times \left(\frac{b}{R} + T_p\right) + \frac{n_b \times b}{R_v}
\]

\[
T_c = n_c \times \left(\frac{b}{R} + T_p\right) + \frac{n_c \times b}{R_v}
\]

\[
T_s = n_s \times \left(\frac{b}{R} + T_p\right) + \frac{n_s \times b}{R_v} + n_s \times T_{proofs}
\]

The first term represents the downloading of the blocks from the existing node where as the second term denotes the validation of the blocks. In a securePrune network, the nodes need to validate the NI-PoE proofs in time \(T_{proofs}\) for each block verification, whereas the number of blocks is less compared to the other two networks.

The bootstrap node, after obtaining its final state from the most recent snapshot and tailchain could acts as a full node to bootstrap the new joining nodes.

VI. RESULTS AND DISCUSSION

Table II lists the values of the parameters used for generating the results in this section. See Table I for a description.

We have conducted an event-driven simulation using python by generating events as per information propagation protocol of bitcoin for propagating a block from miner to reach the entire network. The events are classified as inv - sending a new block hash invitation, getblock - requesting a new block, block - sending a block to its peers and addblock - adding a received block to its local copy of blockchain.

We have simulated for a duration of 70 days (equivalent to 10000 blocks with a block creation rate of \(\lambda = \frac{1}{3600}\) (1 block per every 10 minutes) similar to bitcoin block generation rate. We have chosen 13 nodes as miners with hash rates as per hash distribution shown in [25].

Fig. 4 show the time required for a full node to verify NI-PoE proofs \((\pi_d\) and \(\pi_a\)) with respect to the number of deleted sources of inputs \(|S_d|\) and number of added outputs \(|S_a|\). The verification time is very less \((\approx 0.35\) sec for 100 inputs and 100 outputs) compared to the block-creation time \((600\) sec).

| Parameter | Value |
|-----------|-------|
| \(n\)     | 1000  |
| \(n_p\)   | 8     |
| \(\lambda\) | \(1/600\) blocks/sec |
| \(T_p\)   | 30 msec |
| \(b\)     | 0.25 MB |
| \(R\)     | 10 Mbps |
| \(k\)     | 500   |
| \(\Delta_s\) | 1000 |
| \(R_v\)   | 0.25 Mbps |
| \(T_{proofs}\) | 0.35 sec |

TABLE II:
Parameter values used for simulations
Fig. 4: Verification time of NI-PoE proofs Vs Number of inputs/outputs transactions per block

Fig. 5: Storage comparisons of a nodes in different protocols

Fig. 6: The bootstrapping time of new nodes with respect to Block Height

**Denial-of-Service attack on coinPrune protocol:** There is possible DOS attack on coinPrune by the miners in the network. Since the coinPrune requires a $k$ number confirmations out of the number of blocks in reaffirmation window to prune the blocks prior to the snapshot. If the minimum requirement of $k$ confirmations not attained for any snapshot, then the pruning could postponed to the next reaffirmation window. We chose miners $m_{DoS} = \{1, 7, 8, 10, 12, 13\}$ arbitrarily as DoS attackers with collective hash rate of $0.377 \approx 38\%$ and the other miners participating in the reaffirmations are $m_{reaffirm} = \{2, 3, 4, 5, 6, 9, 11\}$ have collective hash rate of $0.623 \approx 62\%$. We chose 300 (60% of size of reaffirmation window) confirmations out of reaffirmation window of size 500 blocks. The coinPrune shows the pruning at block heights 3500 and 7500. In this case, the pruning of the coinPrune node postponed a duration of 1000, 2000 blocks due to DoS attack by the above mentioned miners even though the number of confirmations (300) chosen are less than the number of confirmations (500) chosen for securePrune block.

For values given in TABLE II the simulation results in Fig. 5 show the maximum storage of securePrune node is approximately 400 MiB ($\Delta_s + k \times b$) for a block size of 0.25 MiB, while the size of the bitcoin full node increases with block height. The results show that 85% reduction in the the storage space of a securePrune node compared to bitcoin nodes.

Fig. 6 show the time required for a bootstrapping node to synchronize with the existing nodes in the network. We hard-coded the block validation rate $R_v$ (depends on the processing speed of a node) and proofs verification time $T_{proofs}$ (from Fig. 4) in the simulation. The synchronization time linearly proportional to the number of blocks present in the chain at the time of joining a new node. The difference in synchronization time of new nodes in coinPrune and securePrune after pruning is due to the extra time ($T_{proofs}$) required for a new node in securePrune to verify the NI-PoE proofs. Fig. 6 show a significant reduction in synchronization time for a new node joining securePrune network compared to nodes joining the other two protocols.

Note: The results shown in Fig. 5 and Fig. 6 are obtained for different runs of simulations.
VII. CONCLUSION AND FUTURE WORK

In this paper, we show the periodic and secure pruning of the blocks prior to a certain block height based on the RSA accumulators. We proposed algorithms for generation of a block and validation of the block using NI-PoE proofs and accumulator state for securing the state of the blockchain along with transactions of the blocks. Through simulation results, we show the reduction in the storage space of a node in the proposed protocol which in turn reduce the synchronization time required to bootstrap a new node.

In future, we explore the exchanging of a snapshot from an existing node during the bootstrap of a new node while the state of the serving node changes with the creation of new blocks. We also consider the trade-off between the block generation, verification with efficient NI-PoE proofs and number of transactions in the block.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2009. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf
[2] ethereum/wiki, “A next-generation smart contract and decentralized application platform,” 2015. [Online]. Available: https://github.com/ethereum/wiki/wiki/WhitePaper
[3] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” CoRR, vol. abs/1801.10228, pp. 1–15, 2018. [Online]. Available: http://arxiv.org/abs/1801.10228
[4] MedicalChain. (2018). [Online]. Available: https://medicalchain.com/en/
[5] R. J. Ali Dorri, Salil S. Kanhere and P. Gauravaram, “Blockchain for iot security and privacy: The case study of a smart home,” in IEEE PERCOM Workshop On Security Privacy And Trust In The Internet of Things 2017. Kona, HI, USA: IEEE, March 2017, pp. 618–623.
[6] BitcoinCore, “Bitcoin Full Node,” 2020. [Online]. Available: https://bitcoin.org/en/full-node
[7] S. Delgado-Segovia, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomarti, “Analysis of the bitcoin utxo set,” in Financial Cryptography and Data Security, A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 78–91.
[8] BitcoinCore, “Bitcoin Source Code,” 2020. [Online]. Available: https://github.com/bitcoin/bitcoin
[9] Blockchain, “Blockchain Luxembourg S.A.” 2020. [Online]. Available: https://www.blockchain.com/charts/blocks-size
[10] N. Barić and B. Pfitzmann, “Collision-free accumulators and fail-stop signature schemes without trees,” in Advances in Cryptology — EUROCRYPT ’97, W. Fumy, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 480–494.
[11] D. Boneh, B. Bünz, and B. Fisch, “Batching techniques for accumulators with applications to iops and stateless blockchains,” in Advances in Cryptology – CRYPTO 2019, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 561–586.
[12] D. Perard, J. Lacan, Y. Bachy, and J. Detchart, “Erasure code-based low storage blockchain node,” CoRR, vol. abs/1805.00860, 2018. [Online]. Available: http://arxiv.org/abs/1805.00860
[13] S. Kadhe, J. Chung, and K. Ramchandran, “Sef: A secure fountain architecture for slashing storage costs in blockchains,” CoRR, vol. abs/1906.12140, 2019. [Online]. Available: http://arxiv.org/abs/1906.12140
[14] M. Luby, “Lti codes,” in The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings., 2002, pp. 271–280.
[15] R. K. Raman and L. R. Varshney, “Dynamic distributed storage for scaling blockchains,” CoRR, vol. abs/1711.07617, 2017. [Online]. Available: http://arxiv.org/abs/1711.07617
[16] A. Shamir, “How to share a secret,” Commun. ACM, vol. 22, no. 11, p. 612–613, Nov. 1979. [Online]. Available: https://doi.org/10.1145/359168.359176
[17] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, “Explicit construction of optimal exact regenerating codes for distributed storage,” in Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing, ser. Allerton ’09. IEEE Press, 2009, p. 1243–1249.
[18] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, “Optimal locally repairable and secure codes for distributed storage systems,” IEEE Transactions on Information Theory, vol. 60, no. 1, pp. 212–236, 2014.
[19] BitcoinCore, “Assumed-Valid vlocks,” 2020. [Online]. Available: https://bitcoin.org/en/release/v0.14.0
[20] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, “How to securely prune bitcoin’s blockchain,” in 2020 IFIP Networking Conference (Networking), 2020, pp. 298–306.
[21] J. Camenisch and A. Lysyanskaya, “Dynamic accumulators and application to efficient revocation of anonymous credentials,” in Advances in Cryptology — CRYPTO 2002. M. Yung, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 61–76.
[22] A. Shamir, “On the generation of cryptographically strong pseudorandom sequences,” ACM Trans. Comput. Syst., vol. 1, no. 1, p. 38–44, Feb. 1983. [Online]. Available: https://doi.org/10.1145/357353.357357
[23] M. Rosenfeld, “Analysis of hashrate-based double spending.” CoRR, vol. abs/1402.2009, pp. 1–13, February 2014. [Online]. Available: http://arxiv.org/abs/1402.2009
[24] C. Decker and R. Wattenhofer, “Information propagation in the bitcoin network,” in IEEE P2P 2013 Proceedings. Trento, Italy: IEEE, Sep. 2013, pp. 1–10.
[25] Blockchain, “Hash Rate Distribution,” 2020. [Online]. Available: https://www.blockchain.com/pools