securePrune:Secure block pruning in UTXO based blockchains using Accumulators

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

2021 International Conference on COMmunication Systems & NETworkS (COMSNETS) | 978-1-7281-9127-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/COMSNETS51098.2021.9352892

***Abstract*—In this paper, we propose a secure block pruning 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. In our scheme, the miners periodically release a *snapshot* of the blockchain *state* (*UTXO set*), the other full nodes in the network, securely prune the historical blocks after attaining the required number of confirmations to the *snapshot block*. This is achieved through the modification of the block structure by including a representation for the *state* as an RSA accumulator called *accumulator state* in the block header and proofs of knowledge for deletion/inclusion of the current block's input/output transactions in 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 the bootstrapping cost of the new nodes.**

***Index Terms*—Blockchain, UTXO set, RSA Accumulator, Pruning, NI-PoE Proofs, Bootstrapping.**

I. Introduction

The Blockchain is a revolutionary technology behind the Peer-to-Peer (P2P) networks like bitcoin [1], Ethereum [2] and Hyperledger [3].

The bitcoin blockchain is a P2P network of miners, full nodes and simplified payment verifiation nodes (SPV) [1]. The miners play a key role in generating the blocks through Proof-of-Work (PoW) puzzle [1].

Every full node stores three types of databases in its storage space - total blocks in the blockchain since the genesis block, unconfirmed transactions (Memory pool) and *UTXO set* [4]. The *UTXO set* keep track of 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 validation of transactions in a block through *UTXO set* database. However, running a full node incurs storage costs as the blockchain data grows exponetially [5] with time. The main advantage of storing the all blocks by the full node is to make the bootstrapping nodes to synchronize with the existing network nodes.

In [6], a bootstrapping node skips the script validation of the transactions for parents 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* [7], the authors proposed a method for pruning older blocks by creating a *snapshot* of the *state* at regular intervals, provided the collective reaffirmations to *snapshot* by the miners in a duration of reaffirmation window. However, there is a possibilty of the *Denial-of-Service* (DOS) attack by the miners in reaffirming the *snapshot*. So, there is no gaurantee that pruning will happen at every *reaffirmation window* of a *snapshot* release.

In this paper, we propose a periodic pruning of the historical blocks based on the security confirmations guaranteed by the RSA accumulator [8], [9] of the *UTXO set*, proofs of knowledge for deletion and inclusion of the current block's input and output transactions. We propose a modified PoW and block validation algorithms based on the modified block structure which include *accumulator state* in the block header and Non-Interactive proof of Exponentiation (NI-PoE) proofs [9] for deletion of the *UTXO* sources of the input transactions and inclusion of the new output transations of the current block. Every full node validates a new block using the NI-PoE proofs and the *UTXO set* using the block validation algorithm. The miners release a *snapshot* of the *UTXO set* at regular intervals of time and every full node prunes the historical blocks prior to the *snapshot* based on the number of confirmations to the *snapshot block*.

The simulation results demonstrate 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 describe the system model and notations used in the protocol. Section III describes the preliminaries of the crypto primitives. In Section IV, we decribe the proposed protocol for secure and periodic pruning and synchronization of the bootstrapping nodes. In Section V, we present and discuss the simulation results. Section VI presents the concluding remarks and future works.

II. System Model and parameters

The following functions of the bitcoin protocol [1] are used in *securePrune* protocol.

- $hash(.)$: cryptographic hash function
- $root(.)$: Merkle root of a set of transactions

- $PoW(.)$: Proof-of-Work function
- $validate(.)$: Transaction validation function

A. Overview of the transactions and UTXO set

The full node stores the *UTXO set* in the chainstate database of the Bitcoin core [10]. The database consists of records of key-value pairs [4] . 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* S_i reprersented as

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

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

B. The modified block structure in the proposed protocol

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

$$C_h = (B_0, B_1, \ldots, B_h) \tag{2}$$

where,

$$B_i = \; < H_i, (A'_{i-1}, \pi_d, \pi_a), t_i > \tag{3}$$

Where, The set $t_i = \{tx_1, tx_2, ..., tx_{|t_i|}\}$ represents the set of transactions in block B_i. The block header H_i consists of an extra element called *accumulator state* A_i in addition to the elements of the bitcoin block header.

$$H_i = (h_{i-1}, R_i, nonce, A_i, x) \tag{4}$$

where, $h_{i-1} = hash(H_{i-1})$, R_i is the merkle root of the t_i, $nonce$ 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 tuple (A'_{i-1}, π_d, π_a) results from the state transition of the *UTXO set*. The element π_d denotes the NI-PoE proof for deletion of Set of utxos S_d spent in the new block B_i from the *accumulator state* and π_a denotes the NI-PoE proof for addition of set of output transactions S_a in the new block B_i to be added to the *accumulator state*. The intermediary *accumulator state* A'_{i-1} is the result of state transition after the deletion of the set S_d from A_i.

III. Preliminaries

The following definitions of RSA accumulators [8], [9] are used in our work.

Let $\mathbb{G}$ be a group of unknown order and $g \in \mathbb{G}$, $S_i = \{u_j : j = 1, 2, ..., |S_i|\}$ and $U_j = H_{prime}(u_j)$. Where, $H_{prime}(.)$ is the prime representation function.

The *accumulator state* A_i of block B_i is computed as

$$A_i = g^{\prod_{j=1}^{|S_i|} U_j} \tag{5}$$

The *membership witness* for $u_m \in S_i$ is defined as

$$W_m = g^{\prod_{j=1, j\neq m}^{|S_i|} U_j} \tag{6}$$

While creating a new block, the miner computes new *accumulator state* A_i from A_{i-1} as follows

$$A_i = BatchAdd(BatchDel(A_{i-1}, S_d), S_a) \tag{7}$$

The Non-interactive PoE (NI-PoE) [9] proofs π_d and π_a are generated during the batch updates for the efficient verification without any interaction between prover(miner) and verifier(full node).

Let $S_d = \{u_1, u_2, \ldots, u_{|S_d|}\}$ and $S_a = \{v_1, v_2, \ldots, v_{|S_a|}\}$. *BatchDel* creates an intermediary *accumulator state* A'_{i-1} and π_d.

$$A'_{i-1} = W_{agg} = g^{\prod_{s \in S_{i-1} \setminus S_d} s} \tag{8}$$

where, W_{agg} is an aggregated membership witness of all $u_j \in S_d$ computed by using Shamir Trick [9].

$$\pi_d = NI - PoE(A'_{i-1}, U^*, A_{i-1}) \tag{9}$$

Finally, *BatchAdd* compute A_i and π_a as follows

$$A_i = (A'_{i-1})^{V^*} \tag{10}$$

$$\pi_a = NI - PoE(A'_{i-1}, V^*, A_i) \tag{11}$$

where,

$$U^* = \prod_{u_j \in S_d} H_{prime}(u_j), V^* = \prod_{v_j \in S_a} H_{prime}(v_j) \tag{12}$$

The miner calculates new membership witnesses for the elements of new state S_i through *updateMemWit* function. 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 (6), then the updated membership witnesses for all $s \in S_{i-1} \setminus S_d$ are generated as follows

$$w'_s = ShamirTrick(A'_{i-1}, w_s, U^*, s) \tag{13}$$

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

$$w''_s = (w'_s)^{V^*} \tag{14}$$

The membership witnesses for all $v_j \in S_a$ are calculated as follows

$$w_{v_j} = (A'_{i-1})^{\prod_{v_m \in S_a, v_m \neq v_j} v_m} \tag{15}$$

IV. Secure block pruning protocol

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 *accumulator state* and NI-PoE proofs.

A. Requirements of the securePrune protocol

1) State transition Algorithm: The *UTXO set* of the blockchain is dynamic and changed for every new block addition to the blockchain. Algorithm 1 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_i - new state, S_d, S_a

procedure STATETRANSISTION (S_{i-1}, t_i)
  $S' \leftarrow S_{i-1}$
  for tx in t_i **do**
    $isValid \leftarrow validate(tx)$
    if $isValid$ **then**
      for *input* in *tx* **do**
        $id \leftarrow input[txHash]$
        $S'.delete(u_j[id])$, $S_d.append(u_j[id])$
      end for
      for $output$ in tx **do**
        S'.append($output$), S_a.append($output$)
      end for
    else
      return *False*
    end if
  end for
  $S_i \leftarrow S'$
  return S_i, S_d, S_a
end procedure

2) Modified PoW Algorithm: The modified PoW function for mining a new block is described in Algorithm 2. This PoW funtion includes Accumulator state A_i along with other parameters into the block header for providng immutable blockchain state S_i. It also includes NI-PoE proofs (π_d, π_a) for *deletion* and *addition* of the new set of elements (S_d, S_a) to the state from the current transaction set t_i.

Algorithm 2 The modified *PoW* function for the secure prune protocol

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

procedure SECUREPRUNEPOW(S_{i-1}, C_{i-1})
  for tx in M **do**
    t_i.append(tx)
    if size of $B_i >$ Max Block Size **then**
      break
    end if
  end for
  $S_i, S_d, S_a \leftarrow stateTransation(S_{i-1}, t_i)$
  $A'_{i-1}, \pi_d \leftarrow BatchDel(A_{i-1}, S_d, W)$
  $A_i, \pi_a \leftarrow BatchAdd(A'_{i-1}, S_a)$
  $nonce \leftarrow PoW(H_{i-1}, R_{t_i}, A_i, x)$
  $H_i \leftarrow < H_{i-1}, R_{t_i}, A_i, x, nonce >$
  $B_i \leftarrow < H_i, \pi_d, \pi_a, t_i >$
  $W \leftarrow W'$, $C_i \leftarrow C_{i-1}B_i$
  $W' = updateMemWit(A'_{i-1}, W, S_d, S_a)$
  return C_i
end procedure

Algorithm 3 Block Validation Algorithm

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

procedure VALIDATEBLOCK(S_{i-1}, C_{i-1}, B_i)
  $t_i \leftarrow B_i[t_i]$, $count \leftarrow 0$
  for tx in t_i **do**
    $isValid \leftarrow validate(tx)$
    if not $isValid$ **then**
      return *False*
    end if
    $count \leftarrow count + 1$
  end for
  if $R_i \neq root(t_i)$ **then**
    return *False*
  end if
  if $count == |t_i|$ **then**
    $A'_{i-1}, \pi_d, \pi_a \leftarrow B_i$, $A_{i-1} \leftarrow B_{i-1}[accState]$
    $S_i, S_d, S_a \leftarrow stateTransition(S_{i-1}, t_i)$
    $a \leftarrow NI{-}PoE.Verify(\prod_{s \in S_d} s, A'_{i-1}, A_{i-1}, \pi_d)$
    $b \leftarrow NI - PoE.Verify(\prod_{s \in S_a} s, A'_{i-1}, A_i, \pi_a)$
  end if
  if $a \wedge b$ **then**
    $S_i \leftarrow S'$, $C_i \leftarrow C_{i-1}B_i$
  end if
  return C_i
end procedure

3) Block Validation Algorithm: We defined a validation function in Algorithm 3 to check the validity of A_i, t_i, R_i, π_d and π_a from the present *state* S_{i-1} and the received new block B_i. If B_i is valid, the full nodes append B_i to C_{i-1}, otherwise discard 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 Δ_s blocks called *snapshot interval*. The miners while creating a new block as per the Algorithm 2 at a height $c\Delta_s$ ($c = 1, 2, 3, \dots$) releases the *snapshot* along with the block $B_{p+c\Delta_s}$ created at that particular height. The snapshot conststs of an *indetifier* and a copy of the *state* $S_{p+c\Delta_s}$ (include the unspent transactions of the current block also). The *snapshot identifier* is the *accumulator state* present in the block header of *snapshot* block $B_{p+c\Delta_s}$. The chain subsequent to the snapshot block $B_{p+c\Delta_s}$ is termed as the *tailchain*. The full node follows the Algorithm 3 for validation of a block created during Δ_s (present in the *tailchain*) by verifying the NI-PoE proofs π_d and π_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 as shown in Fig. 1, 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 [1] for deciding the number of confirmations on B_p.

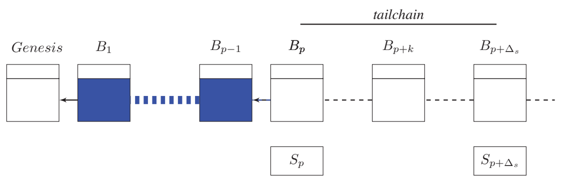

Fig. 1: Overview of *securePrune* protocol: The blue coloured blocks are pruned after attaining a k confirmations to block B_p.

C. Synchronization of the Bootstrapping nodes

The new node joining the network bootstrap in three steps - First, it obtains the most recent *snapshot* with the longest *tailchain*. Second, the new node downloads the entire *headerchain* since the *genesis* block and verifies the validity of the *headerchain*. Third, the node downloads the *tailchain* from its peers and validate all the blocks since the most recent *snapshot* block to obtain its *state*.

Let S_p is the most recent *snapshot* and $\bullet$ denotes the state transition function. If a node joins the network at height h, then the state of the new node at height h is obtained as

$$S_h = S_p \bullet B_{p+1} \bullet \cdots \bullet B_h \quad (16)$$

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

V. Results and Discussion

Table I lists the values of the parameters used for generating the results in this section.

We have conducted an event-driven simulation using python by generating events as per information propagation protocol [11] 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 $\lambda = \frac{1}{600}$ (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 [12].

TABLE I:
Parameter values used for simulations

Parameter	**value**	Description
n	1000	Number of nodes
n_p	8	Number of peers
λ	1/600	Block creation rate
T_p	30 msec	Propagation delay
b	0.25 MB	Block size
R	10 Mbps	Average download bandwidth
k	500	Number of confirmations
Δ_s	1000	Snapshot interval

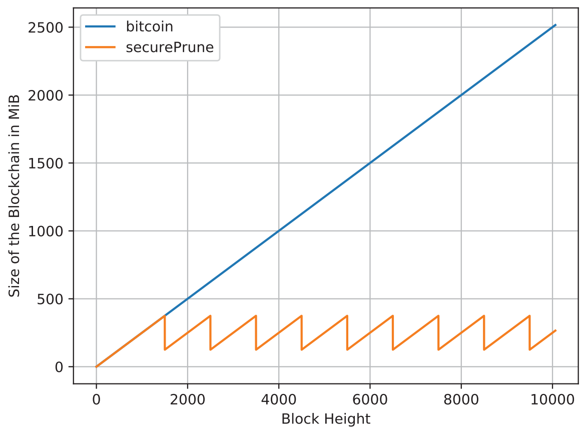

Fig. 2: Storage comparisons of a nodes of bitcoin and *securePrune* protocols

Fig. 2 show the total blockchain size of the nodes with respect to the block height. We have chosen the $\Delta_s = 1000$ blocks between the two consecutive *snapshot*s and $k = 500$ number of confirmations (as per the calculations given in [1] ≈ 462 number of confirmations required for double-spend to succeed by an attacker (with fraction of hash rate $q = 0.45$) with a probability $< 10^{-4}$) for pruning the blocks prior to the *snapshot*. The nodes prune old blocks at height $h = 1000 + 500c$ ($c = 1, 2, 3, \ldots$).

For values given in TABLE I, the simulation results in Fig. 2 show the maximum storage of *securePrune* node is approaximately 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 full nodes. As a result, the synchronization time of a new boot strapping node decreases significantly in *securePrune* network.

VI. 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 the 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 reduces the synchronization time required to bootstrap a new node.

In future, we explore the exchanging of a *snapshot* from an existing node during the bootstrapping process 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 computational complexity of the proposed algorithms and transaction throughput of the *securePrune* network.

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] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "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.

[5] Blockchain, "Blockchain Luxembourg S.A." 2020. [Online]. Available: https://www.blockchain.com/charts/blocks-size.

[6] Bitcoin, "Assumed-Valid vlocks," 2020. [Online]. Available: https://bitcoin.org/en/release/v0.14.0

[7] R. Matzutt, B. Kalde, J. Pennekamp, A. Drichel, M. Henze, and K. Wehrle, "How to securely prune bitcoins blockchain," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 298–306.

[8] 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.

[9] 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.

[10] BitcoinCore, "Bitcoin Source Code," 2020. [Online]. Available: https://github.com/bitcoin/bitcoin

[11] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. Trento, Italy: IEEE, Sep. 2013, pp. 1–10.

[12] Blockchain, "Hash Rate Distribution," 2020. [Online]. Available: https://www.blockchain.com/pools.