Front. Comput. Sci., 2024, 18(2): 182806
https://doi.org/10.1007/s11704-023-2365-9

RESEARCH ARTICLE

CompactChain: an efficient stateless chain for UTXO-model blockchain

B Swaroopa REDDY, T Uday Kiran REDDY (✉)

Department of Electrical Engineering, Indian Institute of Technology Hyderabad, Hyderabad, Telangana 502285, India

Abstract In this work, we propose a stateless blockchain called CompactChain, which compacts the entire *state* of the UTXO (Unspent Transaction Output) based blockchain systems into two RSA accumulators. The first accumulator is called Transaction Output (TXO) commitment which represents the *TXO set*. The second one is called Spent Transaction Output (STXO) commitment which represents the *STXO set*. In this work, we discuss three algorithms: (i) To update the TXO and STXO commitments by the miner. The miner also provides the proofs for the correctness of the updated commitments; (ii) To prove the transaction's validity by providing a membership witness in TXO commitment and non-membership witness against STXO commitment for a coin being spent by a user; (iii) To update the witness for the coin that is not yet spent; The experimental results evaluate the performance of the CompactChain in terms of time taken by a miner to update the commitments and time taken by a validator to verify the commitments and validate the transactions. We compare the performance of CompactChain with the existing state-of-the-art works on stateless blockchains. CompactChain shows a reduction in commitments update complexity and transaction witness size which inturn reduces the mempool size and propagation latency without compromising the system throughput (Transactions per second (TPS)).

Keywords stateless blockchain, RSA Accumulator, STXO commitment, TXO commitment, UTXO, Non-interactive Proof of Exponentiation (NI-PoE), Transactions per second (TPS)

1 Introduction

Blockchain has emerged as a decentralized and trustless technology for both cryptocurrencies and smart contract applications. An anonymous person named Satoshi Nakamoto introduced Bitcoin [1] as a Peer-to-Peer ($P2P$) network that records an append-only immutable ledger using cryptographic digital signatures and hash functions. Blockchain-based smart contract platforms like Ethereum [2], Hyper ledger fabric [3] open the applications of blockchain in many areas like Internet-of-Things (IoT) [4], supply chain management [5], healthcare [6], agriculture [7], energy trading [8], etc.

A transaction is a fundamental entity in the blockchain ledger. It is defined as transferring the ownership of a coin from one party to the other through digital signatures [1]. In a transaction, the destinations of ownership transfer are called *outputs*, and the sources of ownership are called *inputs*. A subset of previous *outputs* is spent as *inputs* of the transaction. The transactions contain multiple *inputs* and *outputs*. The set of the Unspent Transaction Outputs (UTXO) is called the *state* of a blockchain. The *state* changes dynamically as a subset of previous *outputs* (or *inputs*) are being spent, and create new *outputs*.

The blockchain network is jointly maintained by two different types of nodes - miners, and users. The *miners* are the special nodes who invest their computational power to solve the Proof-of-Work (PoW) [1] puzzle to create immutable blocks. The *full nodes* [9] and *miners* in the network act as *validators*. The validators store the complete *UTXO set* (state) [10] to verify the validity of blocks and transactions. So, bitcoin is an example of a UTXO based stateful blockchain. The full nodes store the *UTXO set* in the chainstate directory, a level dB database of the bitcoin core [11]. The validators also store the complete blocks from the genesis to the present block and help new nodes joining the network to synchronize with the blockchain's current state.

The blockchain [12] and *UTXO set* [12] sizes are ever-growing and increase the storage burden on validators and reduce transaction verification performance. Consequently, some users run a Simplified Payment Verification (SPV) node or light client [1]. The SPV nodes rely on the full nodes to verify the transaction by querying the Merkle path from the transaction to the Merkle root listed in the block header. Further increase in the size of the *UTXO set* might exceed the RAM capacity of the validator leads to an increase in the count of the SPV nodes compared to the validator nodes, which underpins the decentralization property of the blockchain system.

Accumulator is a cryptographic primitive that produces a short binding commitment to a set of elements together with short membership/non-membership proofs for any element in the set. The membership witness is a single element, which

Received June 7, 2022; accepted February 1, 2023

E-mail: ee19btech11038@iith.ac.in

allows us to prove the fact that a given element is included in the accumulator. Similarly, the non-membership witness allows us to prove the fact that a given element is not a part of the accumulator. The RSA accumulator [13,14,15] is an accumulator based on strong RSA assumption.

In [16], the authors propose a stateless blockchain for UTXO-model and account-model blockchains using the batching techniques on trapdoor-less RSA accumulator [14,15,17]. In the UTXO-model framework, the block contains a commitment to the latest *UTXO set* called the *accumulator state* or *UTXO commitment*. The accumulator state is verified by the validators using Non-Interactive Proof of Exponentiation (NI-PoE). The users provide the membership witness for their coins being spent in transaction payloads. The miners and fullnodes verify the transactions by checking the membership witnesses against the latest accumulator state. This framework uses batching techniques on RSA accumulators, namely *batchDel* and *batchAdd* to update the commitment for every new block creation. The *batchDel* operation deletes the spent coins or inputs of the transaction from the UTXO commitment. The *batchAdd* operation adds the new coins or outputs to the UTXO commitment. However, the *batchDel* operation aggregates membership witnesses for all inputs of the block into a single membership witness using *shamirTrick*. The *batchDel* is a sequential iterative operation with a complexity of $O(m^2)$. Where m is the number of transactions (assuming single input per transaction). Moreover, it is not possible to exploit parallelism in the computation. Since the complexity of *batchDel* operation is quadratic in time, the efficiency drastically reduces with an increase in the total number of inputs in a block. It severely affects the transaction throughput (TPS). Also, users need to rely on the service providers to update their membership witnesses with every block of transactions.

Minichain protocol proposed in [18] is a light-weight stateless blockchain. Minichain consisting of two commitments - STXO (Spent Transaction Output) commitment, and TXO (Transaction Output) commitment instead of a single UTXO commitment to avoid more complex *batchDel* operation. The STXO commitment is an RSA accumulator to all the spent coins or inputs (called *STXO set*). The TXO commitment is based on the Merkle Mountain Range (MMR) [19] to outputs (called *TXO set*). The validators in Minichain also use the NI-PoE to verify the STXO commitment. While spending a coin, the user provides a non-membership witness (Unspent proof) against STXO commitment and a membership witness (Existence proof) in TXO commitment. The existence proof consists of two Merkle proofs. First, the *coin inclusion proof* to prove the coin is committed to the TMR (Transaction Merkle Root) of the block where the coin is generated. Second, the *TMR inclusion proof* to prove that the TMR is committed to the latest TXO commitment. However, the complexity of the coin inclusion proof and TMR inclusion proof sizes are $O(log_2(m))$ and $O(log_2(L))$. Where L is the length of the blockchain from the genesis block. L is ever-growing and increases network communication latency due to the large membership proof size, which further affects the TPS. This impact is huge in the blockchain networks with high block creation rate.

In this work, we propose a compact stateless blockchain for UTXO-model blockchain by compressing the entire *state* of the blockchain into two RSA accumulators, one each for TXO commitment and STXO commitment. The miner updates the commitments by using two *batchAdd* operations on inputs and outputs. The validators verify both commitments by checking the NI-PoE proofs. The users provide transaction proof consisting of membership witness for a coin in TXO commitment and a non-membership witness against the STXO commitment. The membership witness is also an accumulator excluding the particular coin whose membership is to be proved. Consequently, membership proof is of constant size. The user needs to update the membership and non-membership proofs for their coins with every new block generation to make the proofs compatible with the latest commitments. The validator can efficiently verify the transaction proofs through the RSA group operations.

The main contributions of this work are the following -

- We propose an RSA accumulator for the TXO commitment for a constant sized transaction membership proof in contrast to the ever-growing existence proof size in minichain protocol.
- We implement CompactChain and compare the performance with Boneh's[1)] and Minichain protocols. Comparing to Boneh, CompactChain has improved the efficiency of commitment update from $O(m^2)$ to $O(m)$. Comparing to Minichain, the transaction proof size has improved from $O(log_2(m)) + O(log_2(L)) + O(1)$ to $O(1)$.
- Through simulation results, we show the performance improvement in network communication latency and TPS compared to Minichain due to reduced transaction proof size.

The rest of the paper is organized as follows. We discuss the related work in Section 2. Section 3 provides the preliminaries. In Section 4, we present the system architecture. In Section 5, we discuss the design of CompactChain protocol. Section 6 demonstrates the performance evaluation of CompactChain in comparison with Boneh's and Minichain protocols. In Section 7, we conclude our work and discuss the future directions of research.

2 Related work

The distributed coding theory techniques like erasure-codes [20] and fountain codes [21] have been proposed for storage efficiency of the blockchain node by reducing the storage cost and still contribute to bootstrap a new node joining the network. In Dynamic distributed storage system [22], the blockchain nodes are allocated into dynamic zones and the nodes in each zone store a share of private keys using Shamir's

1) We use the term Boneh's protocol for the stateless blockchain proposed in [10].

secret-sharing [23] for encrypting the block data and apply a distributed storage codes such as [24,25] for reducing the storage cost.

A snapshot-based block pruning technique has been proposed in [26] to prune archived blocks by creating a *snapshot* of the state at regular intervals. A snapshot-based consensus protocol for bitcoin-like blockchains has been proposed in [27], where the miners create a block by providing a non-interactive proofs of storing a subset of the past state snapshots. However, these techniques solve the problem of storing the historical blocks.

A stateless client concept proposed in [28] for Ethereum blockchain, where full nodes only store a state root and miners broadcast witness (a set of the Merkle branches proving the data values in block) along with the block. The validators download and verify these expanded blocks. In [19], the author proposes low-latency delayed TXO commitments based on Merkle Mountain Range (MMR) for committing to the state of all transaction outputs. In order to append a new output requires fewer storage requirements ($log_2(n)$ mountain tips). The added output could not be removed from the MMR, instead updates the status of the spent output. While spending a coin, each transaction would be accompanied by a Merkle proof consisting of a Merkle path to the tip of a tree such that the outputs being spent were still unspent.

EDRAX [29] proposes a stateless transaction validation for UTXO based blockchain using the Sparse Merkel Tree (SMT) [30]. The coin which is being spent also includes a witness of unspent. However, the witness for a transaction being spent depends on the Merkle proof of size $log_2(m)$.

In [16], the authors proposed a stateless blockchain scheme based on the batching and aggregating techniques for the accumulators of unknown group order. In this scheme, each block contains an accumulator, which represents a commitment to the current UTXO set. This commitment is constructed by leveraging the batching techniques for membership and non-membership proofs for the set of transactions included in the block. However, the complexity of this commitment update is of $O(m^2)$. Where m is the number of transactions (assuming single input per transaction). The efficincy in computing the commitment reduces with the increase in number of average transactions per block.

3 Preliminaries

3.1 State of an UTXO based blockchain

In a UTXO-model blockchain design, the *state* or *UTXO set* [10] is a collection of the Transaction Outputs (TXO) that are unspent at a particular moment of time. Whenever a new transaction is created, it consumes a subset of the current *UTXO set* through *inputs* and creates new UTXOs through *outputs*. A validator in a stateful blockchain needs to keep a copy of the *state* as a *chainstate* database to verify the transaction's validity. The validators are required to update the *state* with a block of transactions by deleting the UTXOs associated with inputs from the *UTXO set* and adding new outputs to the *UTXO set*.

Let S_n be the *state* of a blockchain until block height n and $\bullet$ represents a state transition function. When a new block B_{n+1} is created with a set of transactions included in it, then the *state* of the blockchain is

$$\begin{aligned} S_{n+1} &= S_n \bullet B_{n+1} \\ &= (\cdots((S_0 \bullet B_1) \bullet B_2) \bullet \cdots \bullet B_{n+1}). \end{aligned} \tag{1}$$

3.2 Stateless blockchain

In contrast to the stateful blockchain, in stateless blockchain [16,18] a cryptographic commitment like RSA accumulator [16] to the *UTXO set* is stored in every block header. The validators no longer need to store the complete *UTXO set* in their storage to verify the transactions, instead, the witness provided by each user for their transactions are used to verify the transaction's validity against the latest commitment to the *state*. Any miner update the commitment from inputs and outputs of the transactions in the block and output a proof of correctness. The miner propagate the proof to other nodes in the network.

Let C_n be a commitment to the *UTXO set* in block B_n and $\diamond$ be the commitment update function, then the new commitment for the block at height $n+1$ is obtained as

$$C_{n+1} = C_n \diamond B_{n+1}. \tag{2}$$

The steps to run the network in a stateless blockchain are as follows:

1. Users broadcast transactions along with corresponding witnesses to all the nodes.
2. Miners collects new transactions into the block and update the commitment concerning the commitment of the previous block. It also outputs a proof for correctness of the commitment.
3. Each miner works on finding the Proof-of-Work (PoW) for its block.
4. When a miner finds a PoW, it broadcasts the block along with the witnesses to the transactions in the block to other nodes in the network.
5. Validators accept the block only if all the transactions in it are valid based on the transaction witnesses and the proof of correctness to the commitment is verified.
6. Nodes express their acceptance of the block by working on creating the next block in the chain.
7. The owners of the unspent coins update their transaction proofs based on the newly accepted block of transactions in the chain.

3.3 RSA accumulator

3.3.1 Cryptographic assumptions

An RSA accumulator [13,17] is a cryptographic primitive that generates a short commitment to a set of elements with efficient membership and non-membership proofs for any element of the set.

The RSA accumulator requires the generation of a group of unknown order in which strong RSA and root assumption [16] holds, and it is based on the modular exponentiation with an RSA modulus.

Let $\mathbb{G}$ be a group of unknown order, $g \in \mathbb{G}$ be a generator of the group and N be an RSA modulus such that $N = pq$, where p and q are strong primes [31].

Let $GGen$ is an algorithm that generates the above public parameters and $\mu(\lambda)$ is a negligible function in the security parameter λ.

Strong RSA assumption [13,14,16]: Given RSA modulus N (of size λ)) and a random $y \in \mathbb{Z}_N$, then it is computationally hard to find $x \in \mathbb{Z}_N$ and $l > 1$ such that $x^l \equiv y \bmod N$. i.e., for all probabilistic polynomial time (PPT) adversary $\mathcal{A}$

$$Pr\left[x^l = y : \begin{array}{c} \mathbb{G} \leftarrow GGen(1^\lambda), g \in \mathbb{G}; \\ (x,l) \in \mathbb{G} \times \mathbb{Z}_N \leftarrow \mathcal{A}(\mathbb{G}, g) \end{array} \right] \leqslant \mu(\lambda).$$

The elements to be accumulated must be prime numbers in order to be collision-free [13] under strong RSA assumption. Let $S = \{e_1, e_2, \ldots, e_m\}$ be a set of elements to be accumulated, then the accumulator of elements of S is

$$A = g^{x_1 . x_2 . \ldots . x_m}, \tag{3}$$

where

$$x_i = H_{prime}(e_i), \tag{4}$$

where H_{prime} a random hash to prime function [16].

The dynamic accumulator [14] supports the addition or deletion of the elements to the accumulator.

Let $S_{new} = \{e_{m+1}, e_{m+2}, \ldots, e_n\}$ be a set of new elements to be added, then the batch addition of set S_{new} update the accumulator to $A_{new} \leftarrow A^{x^*}$, where $x^* = x_{m+1}.x_{m+2}.\ldots.x_n$.

3.3.2 Membership and non-membership witness

The function membership witness w_i [15], batching of an element $e_i \in S$ is simply an accumulator without the element e_i

$$w_i = g^{\prod_{j=1, j \neq i}^{m} x_j}. \tag{5}$$

The membership of e_i in A is verified by checking

$$(w_i)^{x_i} \overset{?}{=} A. \tag{6}$$

Let $e \notin S$ and $x = H_{prime}(e)$, then non-membership witness u_x of e in S uses the fact that $gcd\left(x, x^* = \prod_{i=1}^{m} x_i\right) = 1$. The Bezout coefficients a and b such that $ax + bx^* = 1$ gives the non-membership witness for e is

$$u_x = (d, b) = (g^a, b). \tag{7}$$

The non-membership witness is verified by checking

$$d^x A^b \overset{?}{=} g. \tag{8}$$

3.3.3 Accumulator security (undeniability)

The universal accumulator [15] is a dynamic accumulator with efficient update of the membership and non-membership proofs. We use the trapdoor less universal RSA accumulator [16,32], as there is no single trusted manager exists in distributed blockchain system and updates are processed in batches.

Let generator $g \in \mathbb{G}$ and $\mu(\lambda)$ is a negligible function. The functions **VerMemWit** and **VerNonMemWit** verify the membership and non-membership witnesses as per the Eqs. (6) and (8) respectively. A dynamic universal accumulator is secure [15,32,16] if, for all probabilistic polynomial time (PPT) adversary $\mathcal{A}$,

$$Pr\left[\begin{array}{c} \mathbb{G} \leftarrow GGen(1^\lambda); g \in \mathbb{G}; (A, x, w_x, u_x) \leftarrow \mathcal{A}(pp, g); \\ \mathbf{VerMemWit}(x, w_x, A) \wedge \mathbf{VerNonMemWit}(x, u_x, A) \end{array} \right] = \mu(\lambda).$$

In other words, it is computationally infeasible to find both membership and non-membership proofs for an element $x \in S$. This statement is equivalent to it is computationally infeasible to find a non-membership witness for $x \in S$ or a membership witness for $x \notin S$.

3.3.4 Non-interactive proof of exponentiation (NI-PoE)

Let $u, w \in \mathbb{G}$, the proof of exponentiation [16,33] in the Group $\mathbb{G}$, when both the prover and verifier are given $(u, w, x \in \mathbb{Z})$ as inputs and prover wants to convince the verifier that $w = u^x$. Given base u, exponent x, and modulus N, the complexity of computing $u^x \bmod N$ is $O(log(x))$ [34]. In PoE protocol, the verifier's work is much less than computing $u^x \bmod N$, when $x \in \mathbb{Z}$ is much larger than $|\mathbb{G}|$. The PoE can be made non-interactive using the Fiat-Shamir heuristic [35].

```
NI-PoE [16]
........................................................................
Public known $\{x, u, w : u^x = w\}$
# Prove NI-PoE
1: procedure $ProveNI\text{-}PoE(x, u, w)$
2:     $l \leftarrow H_{prime}(x, u, w)$
3:     $q \leftarrow \lfloor \frac{x}{l} \rfloor$
4:     return $Q \leftarrow u^q$
5: end procedure

# Verify NI-PoE
1: procedure $VerifyNI\text{-}PoE(x, u, w, Q)$
2:     $l \leftarrow H_{prime}(x, u, w)$
3:     $r \leftarrow x \bmod l$
4:     $Q^l u^r \overset{?}{=} w$
5: end procedure
```

4 System architecture

We propose a stateless blockchain that is closely built on Boneh's work [16] and Minichain [18]. The parameters used in our framework are listed in Table 1. We stick to Minichain's model with two commitments for *TXO set* and *STXO set* and address the following limitations of the Minichain protocol.

First, the size of a transaction witness consisting of existence and unspent proofs for a coin being spent depends on the length of the blockchain L and the maximum number of transactions m in the block with a complexity of $O(log_2(m)) + O(log_2(L)) + O(1)$. The length of the chain L is ever-growing, and if a coin is not spent for a sufficiently long duration, then the size of the witness is very large. Second, the information being propagated in the network consists of a block and witnesses for transactions associated with that block. Since the witness size depends on L, it affects the end-to-end propagation latency of a block and limits the TPS.

In this work, we propose the RSA accumulator for TXO commitment to address the above limitations. The architecture of the proposed stateless blockchain is shown in Figs. 1 and 2.

TXO commitment To reduce the size of the existence proof of a coin in the Minichain protocol and further to improve the

Table 1 Parameters used in CompactChain

Symbols	Description
B_n	Block at height n
T_n	Set of all transactions in B_n
TXO set	Set of all transaction outputs
STXO set	Set of all spent transaction outputs
TXO_n	Transaction outputs (TXOs) in B_n
$STXO_n$	Spent Transaction outputs (STXOs) in B_n
$TXO_{k:n}$	All TXOs from block B_k to block B_n
$STXO_{k:n}$	All STXOs from block B_k to block B_n
TXO_C_n	Commitment to *TXO set* in block B_n
$STXO\_C_n$	Commitment to *STXO set* in block B_n
Π_{TXO}	NI-PoE proof for TXO_C_n
Π_{STXO}	NI-PoE proof for $STXO\_C_n$
$w_n(x)$	Membership witness to prove $x \in$ *TXO set*
$u_n(x)$	Non-membership witness to prove $x \notin$ *STXO set*

efficiency of the network communication, we propose replacing MMR-based TXO commitment with an RSA accumulator. We use batch addition to update the TXO commitment in every block by adding a set of new outputs in T_n of the block B_n to the TXO commitment in the previous block. When a user wants to spend a coin, he provides a membership witness that specifies the coin included in the TXO commitment. The witness is a single RSA group element, which can be verified using single modular exponentiation.

STXO commitment The STXO commitment is also an RSA accumulator to represent the *STXO set* in the block header. The user provides a non-membership witness against the latest STXO commitment to prove that the coin is not yet spent. When a new block is created in the network, the user who did not spend the coin updates the non-membership witness using spent coins in the block.

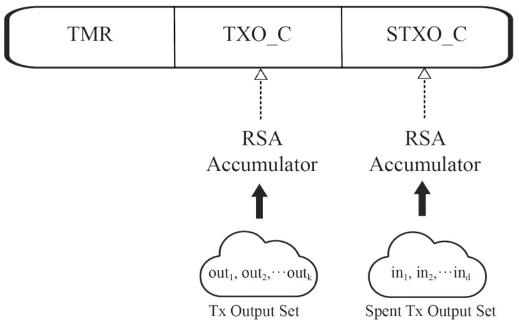

Fig. 1 Block header composition mainly contains two RSA accumulators - TXO_C (TXO commitment) and $STXO\_C$ (STXO commitment)

STXO cache and witness height When the commitments of the blockchain changes before processing a transaction, the user needs to update the membership and non-membership proofs of the coin being spent, which is very unfriendly to the users. We use the STXO cache introduced by Minichain, which contains the STXOs of the latest M blocks. The validator can still verify the non-membership proof is valid by simply querying the STXO cache without updating the proof to the latest STXO commitment by the user.

We introduce the Witness height (h) to specify the height of the block till where the membership and non-membership

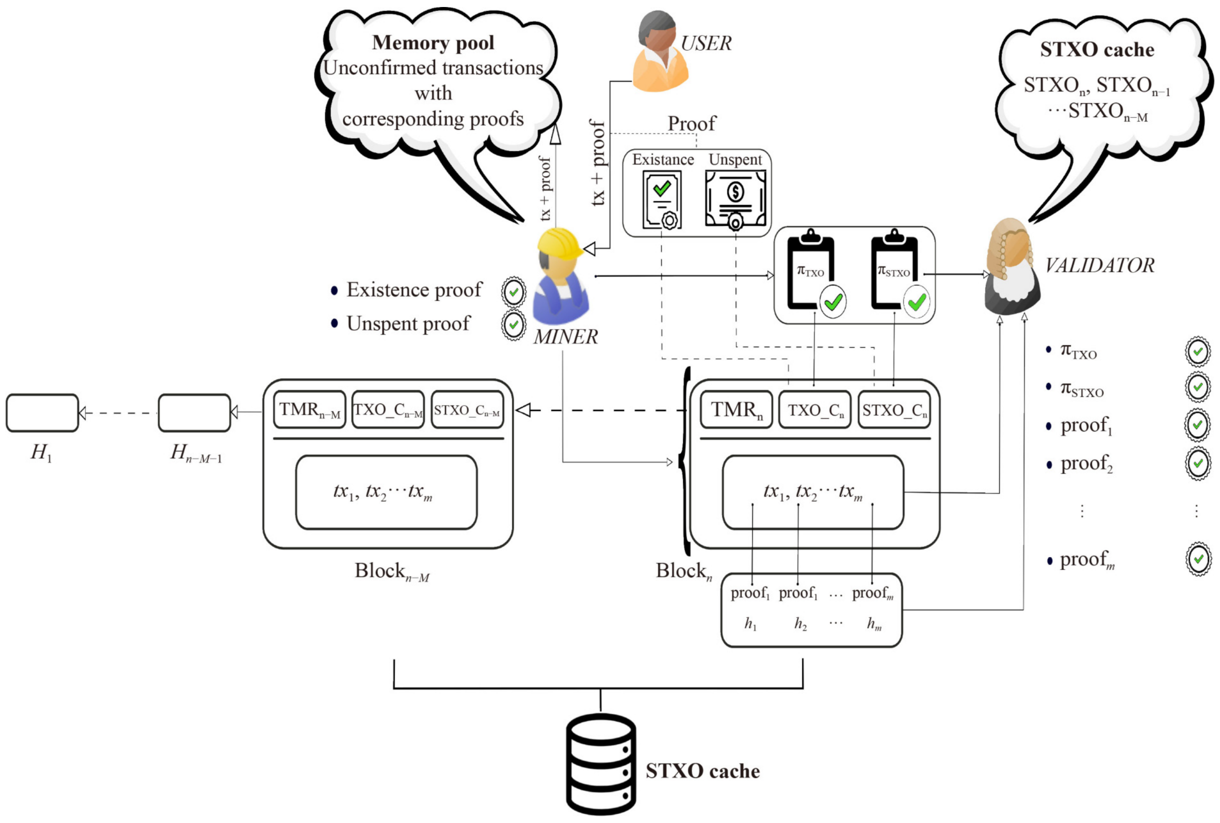

Fig. 2 Block validation procedure

proofs got updated so that the user need not update and resubmit the proofs with every new block creation (or change of the state) after submission of the transaction. While submitting proofs for a coin being spent user provides the witness height h. The validator can verify the membership and non-membership proofs against the TXO and STXO commitments of the block at a height equal to h such that $(n-M) \leqslant h \leqslant n$. Where n is the latest block height, and M represents the length of the STXO cache.

5 CompactChain design

In this section, we discuss the design of the CompactChain protocol - TXO commitment, STXO commitment, witness height, STXO cache, update commitments, transaction witness generation and verification and, finally, update witness. Let λ be a security parameter and TXO_C_n and $STXO\_C_n$ commitments to TXO and STXO sets in n^{th} block. The tuple $(w_n(x), u_n(x))$ denotes the membership and non-membership proofs of a coin x in TXO and STXO sets computed/updated till the block at height n. For brevity, we have omitted mod N from every modular exponentiation operation in the RSA group with modulus N.

5.1 Update TXO and STXO commitments

The block header in the proposed protocol contains two RSA accumulators called TXO_C and $STXO\_C$ to accumulate TXO and STXO sets. The RSA accumulator is cumulative. Thereby, the miner updates the commitments for every block using the commitments from the previous block. The functions to update TXO and STXO commitments are described in Algorithm 1.

The commitments update algorithm takes *inputs*, and *outputs* of a block are input parameters. It returns updated commitments and proof of correctness for the updated commitments using the RSA accumulator's batching techniques. For instance, TXO_C_{n-1} and $STXO\_C_{n-1}$ are the latest commitments to TXO and STXO sets. While generating a new block B_n, the miner computes the new accumulators TXO_C_n and $STXO\_C_n$. Firstly, the $UpdateTXO\_C$ function takes the TXOs (*outputs* of all transactions) from T_n and calculate the prime representatives of each *output* in TXOs using H_{prime} function and computes the product of all prime representatives denoted as p. Finally, the modular exponentiation of base TXO_C_{n-1} with exponent p updates the TXO commitment to TXO_C_n.

Similarly, the $UpdateSTXO\_C$ function takes the STXOs (*inputs*) from T_n as input and update the STXO commitment to $STXO\_C_n$.

The miner also generates NI-PoE proofs Π_{TXO} and Π_{STXO} for both accumulators using the *ProveNI-PoE* function to show that the commitments are updated correctly. The validators need not compute the updated commitments, instead, they can verify the proofs generated by miner using *VerifyComm* function. The products calculated in the *VerifyComm* function are much larger, and recalculating the updated commitments is inefficient to the validator. So, the *VerifyNI-PoE* function used in the *VerifyComm* improves the verification efficiency of the validator.

Algorithm 1 Commitments update algorithm

```
#System Initialization
1: procedure $Setup(\lambda)$
2:     $\mathbb{G} \leftarrow GGEN(\lambda)$
3:     $g \xleftarrow{\$} \mathbb{G}$
4:     $TXO\_C_0 \leftarrow g$
5:     $STXO\_C_0 \leftarrow g$
6:     return $TXO\_C_0, STXO\_C_0$
7: end procedure

#Update $TXO\_C$ function
1: procedure $UpdateTXO\_C(TXO\_C_{n-1}, T_n)$
2:     $p = 1$
3:     for output in $T_n.Outputs$ do
4:         $p \ *= \ H_{prime}(output)$
5:     end for
6:     $TXO\_C_n \leftarrow (TXO\_C_{n-1})^p$
7:     $\Pi_{TXO} \leftarrow ProveNI\text{-}PoE(TXO\_C_{n-1}, p,$
8:                 $TXO\_C_n)$
9:     return $TXO\_C_n, \Pi_{TXO}$
10: end procedure

#Update $STXO\_C$ function
1: procedure $UpdateSTXO\_C(STXO\_C_{n-1}, T_n)$
2:     $p = 1$
3:     for input in $T_n.Inputs$ do
4:         $p \ *= \ H_{prime}(input)$
5:     end for
6:     $STXO\_C_n \leftarrow (STXO\_C_{n-1})^p$
7:     $\Pi_{STXO} \leftarrow ProveNI\text{-}PoE(STXO\_C_{n-1}, p, STXO\_C_n)$
8:     return $STXO\_C_n, \Pi_{STXO}$
9: end procedure

#Verify updated commitments
1: procedure $VerifyCom(T_n, TXO\_C_{n-1}, STXO\_C_{n-1},$
   $TXO\_C_n, STXO\_C_n, \Pi_{TXO}, \Pi_{STXO})$
2:     $p_1 = 1, p_2 = 1$
3:     for output, input in $T_n$ do
4:         $p_1 \ *= \ H_{prime}(output)$
5:         $p_2 \ *= \ H_{prime}(input)$
6:     end for
7:     $b_1 \leftarrow VerifyNI\text{-}PoE(TXO\_C_{n-1}, p_1, TXO\_C_n, \Pi_{TXO})$
8:     $b_2 \leftarrow VerifyNI\text{-}PoE(STXO\_C_{n-1}, p_2, STXO\_C_n,$
9:         $\Pi_{STXO})$
10:    return $b_1 \wedge b_2$
11: end procedure
```

5.2 Transaction witness generation & verification

The TXO commitment combines all the generated coins (*outputs*) and, the STXO commitment combines all the spent coins (*inputs*). While spending a coin, the user submits a proof along with the transaction to show the validity of the coin. The transaction proof consisting of two parts - First, the proof for the coin's existence in *TXO set* and the unspent proof for the non-existence of the coin in the *STXO set*.

The existence proof indicates the coin is generated some time before and is a part of the *TXO set* represented by TXO commitment. Similarly, the unspent proof indicates the coin is not yet spent such that it is not a member of the *STXO set* represented by STXO commitment. In other words, the user needs to generate membership proof corresponding to the latest TXO commitment and non-membership proof against the latest STXO commitment. We stick to a procedure that a user generates transaction witness using the TXOs and STXOs of the block where the coin is generated and apply subsequent witness update for every new block of transactions.

5.2.1 Existence proof (membership witness in *TXO set*)

The existence proof of a coin in the *TXO set* is a membership witness corresponding to the latest TXO commitment. The user generates the proof using the *CreateMemWit* function and the validator verify it using the *VerifyMemWit* function as shown in Algorithm 2.

CreateMemWit function Suppose a coin x_k is generated at block B_k, then $x_k \in$ *TXO set* and accumulated to TXO_C_k. The user constructs the membership witness $w_k(x_k)$ for $x_k \in$ *TXO set* as follows - Firstly, the function collects all the TXOs from TXO_k excluding x_k and computes their prime representatives. Then, it computes the product of all these prime representatives, denoted as p. Finally, the modular exponentiation of the base TXO_C_{k-1} with exponent p generates the witness $w_k(x_k)$.

VerifyMemWit function Suppose $w_k(x_k)$ is a membership witness of the coin $x_k \in$ *TXO set*. Any validator verifies the membership witness by checking $(w_k(x_k))^t \overset{?}{=} TXO\_C_k$, where $t = H_{prime}(x_k)$.

5.2.2 Unspent proof (Non-membership witness in *STXO set*)

The unspent proof of a coin is a non-membership witness corresponding to the latest STXO commitment. The functions *CreateNonMemWit* and *NonMemWitVerify* are used to generate and verify proof for a new coin, as shown in Algorithm 2. **CreateNonMemWit function** Suppose a coin x_k is generated at block B_k and $x_k \notin STXOset$, the user constructs a non-membership witness $u_k(x_k)$ of x_k corresponding to $STXO\_C_k$ as follows. The function collects all the STXOs from $STXO_k$ and computes their prime representatives. Then, it calculates the product of all these prime representatives denoted as p. Since $gcd(t, p) = 1$, where $t = H_{prime}(x_k)$ the unspent proof $u_k(x_k) = (d = (STXO\_C_{k-1})^a$, b). Where a and b are Bezout's coefficients of t and p such that $at + bp = 1$. **VerifyNonMemWit function** Suppose $u_k(x_k)$ is a non-membership witness of $x_k \notin$ *STXO set* computed by a user with the *STXO set* till block B_k. Any validator can verify it by checking $d^t(STXO\_C_k)^b \overset{?}{=} STXO\_C_{k-1}$ as

$$d^t(STXO\_C_k)^b = (STXO\_C_{k-1})^{at}(STXO\_C_{k-1})^{bp}$$
$$= STXO\_C_{k-1}.$$

Algorithm 2 Witness generation and verification algorithm

```
#Prove membership in $TXO\_C_k$ for $x_k$ generated in $B_k$
1: procedure $CreateMemWit(x_k, TXO_k, TXO\_C_{k-1})$
2:     for $txo$ in $TXO_k$ do
3:         if $txo \neq x_k$ then
4:             $p* = H_{prime}(txo)$
5:         end if
6:     end for
7:     $w_k(x_k) \leftarrow (TXO\_C_{k-1})^p$
8:     return $w_k(x_k)$
9: end procedure

#Verify Membership Witness for $x_k$ generated in $B_k$
1: procedure $VerifyMemWit(w_k(x_k), x_k, TXO\_C_k)$
2:     $t \leftarrow H_{prime}(x_k)$
3:     $b \leftarrow ((w_k(x_k))^t == TXO\_C_k)$
4:     return $b$
5: end procedure

#Prove non-membership against $STXO\_C_k$ for $x_k$ generated in $B_k$
1: procedure $CreateNonMemWit(x_k, STXO_k, STXO\_C_{k-1})$
2:     for $stxo$ in $STXO_k$ do
3:         $p* = H_{prime}(stxo)$
4:     end for
5:     $a, b \leftarrow Bezout(H_{prime}(x_k), p)$
6:     $d \leftarrow (STXO\_C_{k-1})^a$
7:     return $u_k(x_k) \leftarrow (d, b)$
8: end procedure

#Verify Non-Membership Witness $u_k(x_k)$ for $x_k$ generated in $B_k$
1: procedure $VerifyNonMemWit(STXO\_C_k, STXO\_C_{k-1}, x_k, u_k(x_k))$
2:     $t \leftarrow H_{prime}(x_k)$ and $d, b \leftarrow u_k(x_k)$
3:     return $(d^t(STXO\_C_k)^b == STXO\_C_{k-1})$
4: end procedure
```

5.3 Transaction witness update

When a user wants to spend a coin, he must submit existence and unspent proofs to show the coin's validity. The submitted proofs might expire due to a delay in network communication, or the miners did not process the transaction in the latest created blocks. In order to bound proofs with the latest commitments, the user must update and resubmit the transaction and witness until the transaction is processed. To address this problem, we require two essential tools. First, the user needs to specify the height of the block h called witness height, where the proofs and commitments are bounded together. The validator checks the membership and non-membership proofs corresponding to TXO_C_h and $STXO\_C_h$. Second, each validator needs to store the STXOs of the latest M blocks (STXO cache) in their database to process the transaction in the subsequent M blocks. If the latest block height is greater than h, the validator checks that the transaction inputs are consumed in the future blocks by simply querying the STXO cache to avoid the double spending of the coin.

The most recent witness updated coins can be further updated by the users themselves using the *UpMemWit* and *UpNonMemWit* functions as shown in Algorithm 3. The witness update is independent of the size of the sets *TXO set* and *STXO set*, which makes a user no need to store the TXO and STXO sets with them.

UpMemWit function Let $w_n(x_k)$ be the membership witness of the coin x_k generated in block B_k. The user can update till B_n $(n > k)$ using *UpdateMemWit*. Firstly, the function gets all the TXOs from block height k+1 to n, denoted as $TXO_{k+1:n}$ and converts them to prime representatives. Then, it computes the product p of all these prime representatives and returns $w_n(x_k)$ using a single modular exponentiation. To verify the updated witness, one check that $(w_n(x_k))^t \overset{?}{=} TXO\_C_n$ (Where, $t = H_{prime}(x_k)$), which holds as

$$(w_n(x_k))^t = (w_k(x_k))^{pt} = (TXO\_C_k)^p = TXO\_C_n.$$

UpNonMemWit function Let $u_k(x_k) = (d, b)$ be non-membership witness of the coin x_k generated in block B_k. The user can update till B_n $(n > k)$ using *UpdateNonMemWit*. Firstly, the function gets all the STXOs from block height k+1 to n are denoted as $STXO_{k+1:n}$ and convert them to prime

Algorithm 3 Witness update algorithm

```
#Update membership proof from block $k$ to $n$
1: procedure $UpMemWit(x_k, w_k(x_k), TXO_{k+1:n})$
2:     for $txo$ in $TXO_{k+1:n}$ do
3:         $p* = H_{prime}(txo)$
4:     end for
5:     return $w_n(x_k) \leftarrow (w_k(x_k))^p$
6: end procedure

#Update non-membership proof from block $k$ to $n$
1: procedure $UpNonMemWit(x_k, u_k(x_k), STXO_{k+1:n})$
2:     for $stxo$ in $STXO_{k+1:n}$ do
3:         $p* = H_{prime}(stxo)$
4:     end for
5:     $a_0, b_0 \leftarrow Bezout(H_{prime}(x_k), p)$
6:     $r \leftarrow a_0 b, d' \leftarrow d\,(STXO\_C_k)^r$
7:     return $u_n(x_k) \leftarrow (d', b_0 b)$
8: end procedure
```

representatives. Then, it calculates the product of all these prime representatives, denoted as p. It calculates Bezout coefficients of $t = H_{prime}(x_k)$ and p as a_0 and b_0. Finally, the updated non-membership witness $u_n(x_k) = \left(d', b'\right)$. To verify it, one check that, $d'^t (STXO\_C_m)^{b'} \overset{?}{=} STXO\_C_{k-1}$, which holds as,

$$\begin{aligned} d'^t (STXO\_C_n)^{b'} &= d^t (STXO\_C_k)^{rt} (STXO\_C_n)^{b_0 b} \\ &= d^t (STXO\_C_k)^{a_0 bt} (STXO\_C_k)^{b_0 bp} \\ &= d^t (STXO\_C_k)^{b(a_0 t + b_0 p))} \\ &= d^t (STXO\_C_k)^{b} = STXO\_C_{k-1}. \end{aligned}$$

5.4 Security analysis

In this section we describe the attacker model for double-spend attack of a transaction and analyse the security of the CompactChain agianst this atatck.

5.4.1 Attacker model

We describe the two attacker models on CompactChain construction as follows:

1. **Double-spend attack** Let a coin x is generated in block B_i as an output of one of the transactions in B_i, i.e., $x \in T_i.Outputs$. x is odd prime. Now, the coin $x \in$ *TXO set* and $x \notin$ *STXO set*. The owner of the coin x computes the transaction witness $W_x = (w_i(x), u_i(x))$ for x from TXO_i and $STXO_i$ using Algorithm 2. Let the updated witness after the creation of the block B_n is $W'_x = (w_n(x), u_n(x))$. When the user wants to spend the coin x in transaction *tx* with x as one of the *inputs* in *tx*, he submits W'_x along with *tx*. Let $n < h < n + M$ such that $tx \in B_h$. Since $x \in STXO_h$, then $x \in$ *STXO set*. i.e., the coin has been spent in block B_h. Suppose, a probabilistic polynomial time (PPT) adversary $\mathcal{A}_1$ wants to double-spend the coin x by including it in tx' (to double-spend the coin which was already spent in transaction *tx*). Let $k > h$ be the block height at which $\mathcal{A}_1$ computes the witness for tx'. The membership proof $w_k(x)$ is easy to prove since $x \in$ *TXO set* and $TXO\_C_k = (w_k(x))^x$. So, the adversary $\mathcal{A}_1$ needs to prove a non-membership witness $u_k(x)$ for coin $x \in$ *STXO set*.
2. For any arbitrary coin $y \notin$ *TXO set*, a PPT adversary $\mathcal{A}_2$ tries to convince the validators of the CompactChain by providing the witness $W_y = (w_k(y), u_k(y))$ at a height k.

In both cases, the adversary to succeed, need to prove the membership and non-membership proofs simultaneously in each of the commitments $STXO\_C$ (Case 1) and TXO_C (Case 2) which is computationally hard as per the undeniability property of the RSA accumulator as per Section 3.3.3.

Theorem 5.1 The CompactChain construction is secure under the strong RSA assumption.

Proof Case 1 (double-spend attack): the adversary $\mathcal{A}_1$ returns a witness tuple $W_x = (w_k(x), u_k(x))$ for $x \in$ *TXO set* and already spent before height k (i.e., x is already included in the commitment $STXO\_C_k$) such that *VerifyMemWit* $(w_k(x), x, TXO\_C_k) = 1$ and *VerifyNonMemWit*$(STXO\_C_k, STXO\_C_{k-1}, x, u_k(x)) = 1$. So, $(d^x . STXO\_C_k)^b = STXO\_C_{k-1}$.

We construct an Algorithm $\mathcal{B}_1$ to break the strong RSA assumption by invoking $\mathcal{A}_1$. $\mathcal{B}_1$ parses $(d, b) \leftarrow u_k(x)$, $w_k \leftarrow w_k(x)$. $\mathcal{B}_1$ compute $w = (STXO\_C_{k-1})^{\prod_{p \in TXO_k, p \neq x} p}$ and $Z = d.w_k^b$. Then, $Z^x = d^x.w^{xb} = d^x.(STXO\_C_k)^b = STXO\_C_{k-1}$, which contradicts the strong RSA assumption.

Case 2 The adversary $\mathcal{A}_2$ returns a witness tuple $W_y = (w_k(y), u_k(y))$ for $y \notin$ *TXO set* such that *VerifyMemWit* $(w_k(y), y, TXO\_C_k) = 1$ and *VerifyNonMemWit*$(STXO\_C_k, STXO\_C_{k-1}, y, u_k(y)) = 1$. So, from these $w_k^y = TXO\_C_k = (TXO\_C_{k-1})^{\prod_{x \in TXO_k} x}$.

We construct an Algorithm $\mathcal{B}_2$ to break the strong RSA assumption by invoking $\mathcal{A}_2$. The Algorithm $\mathcal{B}_2$ parses $w_k \leftarrow w_k(y)$ and compute the following:

1. Compute $x^* = \prod_{x \in TXO_k} x$.
2. Compute a and b such that $a.y + b.x^* = 1$.
3. Compute $Z = w_k^b.(TXO\_C_{k-1})^a$.

Then

$$\begin{aligned} Z^y &= w_k^{by} (TXO\_C_{k-1})^{ay} \\ &= (TXO\_C_{k-1})^{ay + bx^*} = TXO\_C_{k-1}, \end{aligned}$$

which contradicts the strong RSA assumption. □

6 Performance evaluation

In this section, we discuss the performance of CompactChain protocol by comparing it with Boneh's [16] and Minichain[2] [18] protocols. We test the performance of commitments update by miners, transaction witness generation, updation by users. We also test the verification of the NI-PoE proofs for commitments, and transaction witnesses by validators. We also examine the proof size and test performance of the

[2] $Minichain^+$ in [14] is considered as Minichain.

propagation latency of a block in association with transaction throughput. The results shown are averaged over 10 iterations. We have implemented our experiments based on the implementation of [36] in C++. We use the RSA modulus of 3072-bits, 128-bit prime representative and 256-bit Merkle root. We run all our experiments for miner, validator, and user tasks on a machine equipped with Intel(R) Core(TM) $i5-8250U$ CPU @ 1.60 GHz processor and 8 GB of RAM. Our experiments are available at [37].

6.1 Theoretical comparisons

We discuss the theoretical comparison of CompactChain with Boneh's and Minichain protocols. The comparisons are shown in Table 2. All these algorithms depend on the number of coins generated k (*outputs*), the number of coins consumed d (*inputs*), and the number of transactions m in the block.

6.2 Commitments update and verification

In this section, we test the performance of Commitment update and verification discussed in Algorithm 1 and compare with the accumulator updates in Boneh's and Minichain protocols. While creating a new block, the miner must update the commitments due to new TXOs and STXOs.

Fig. 3 shows the comparison of the time consumed for updating the commitments in Boneh's work, Minichain and CompactChain concerning the number of transactions m (we are assuming $k = d$). Boneh's accumulator update consumes much larger time than Minichain and CompactChain due to the $O(m^2)$ complexity of the *batchDel* operation used for accumulator update. For 1000 transactions, Boneh's accumulator update consumes ≈ 240 s, whereas Minichain and CompactChain took ≈ 1 s. The Minichain requires one *batchAdd* operation for d number of spent coins for updating $STXO\_C$. Minichain also updates TXO_C by adding a new TMR in MMR tree.

In the CompactChain, two *batchAdd* operations are required, one each for $STXO\_C$ (d number of spent coins) and TXO_C (k number of generated coins) updates. The *batchAdd* depends on the generation of prime representatives for each element to be accumulated. While creating a new block, d new elements add to $STXO\_C$, and k elements add to TXO_C.

To improve the efficiency of our implementation, we performed computation of prime representatives in parallel as each element is independent of others. Then the computation of updating the commitments TXO_C and $STXO\_C$ are performed in parallel as the set of elements to be added to each commitment are independent. We also exploited parallelism in implementing the Minichain. The slight bias in time taken for commitments update between CompactChain and Minichain is due to additional computation time for calculating prime representatives of k elements added to TXO_C. In both Minichain and CompactChain, the time to update the commitments increases linearly with the transaction count. Figure 4 shows the time taken by a validator for verifying the commitment updates in Boneh's work, Minichain and CompactChain. The verification depends on the computation of the prime representatives and the product of the prime representatives. In Minichain, the validator needs to verify a NI-PoE proof for one RSA accumulator, and Merkle proof for MMR peaks update. Whereas Boneh's work and CompactChain validator verify two NI-PoE proofs.

The validator verifies each NI-PoE proof independently with a constant number of group operations. We have exploited parallelism in commitments verification similar to commitments update. The verification time increases with an increase in transaction count shown Fig. 4.

6.3 Transaction witness generation and verification

The validator needs to verify the existence and unspent proofs

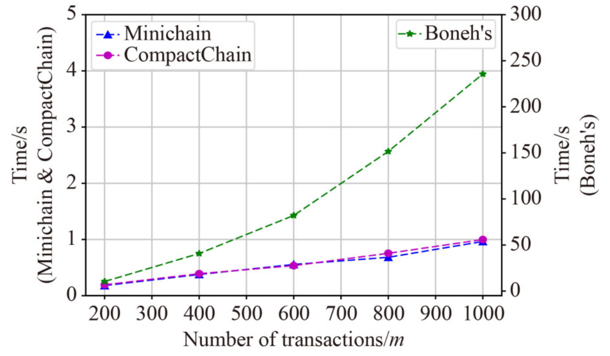

Fig. 3 Performance of the commitments update

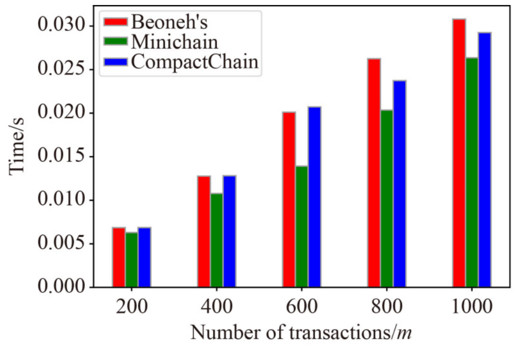

Fig. 4 Performance of the commitments verification

Table 2 Comparison of CompactChain with Boneh's and Minichain protocols

	Boneh's	Minichain	CompactChain
Accumulator type	RSA(UTXO)	MMR(TXO) + RSA(STXO)	RSA(TXO) + RSA(STXO)
Commitment Update	$O(m^2)$	$O(m)$	$O(m)$
Tx Proof generation	$O(k+d)$	$O(log(m))+O(log(L))+O(d)$	$O(k+d)$
Tx Proof verification	$O(1)$	$O(log(m))+O(log(L))+O(1)$	$O(1)$
Tx Proof size	$O(1)$	$O(log(m))+O(log(L))+O(1)$	$O(1)$
Tx Proof update	$O(k+d)$	$O(d)$	$O(k+d)$

of a transaction in both Minichain and CompactChain. The unspent proof is the same in both, but the existence proof is different in size and structure. In Minichain, the validator needs to verify two Merkle proofs - transaction inclusion proof in TMR, and TMR inclusion proof in the latest TXO commitment. We assume the length of the chain $L = 2^{20}$ and $m = 2^{10}$. In CompactChain, the existence proof is an RSA group element, and the validator verifies it by single modular exponentiation. In Boneh's protocol, the validator verifies a single unspent proof (membership in the UTXO commitment) of the transaction's input in the latest commitment.

Table 3 shows the comparison of the proof size and verification time. The results show that the proof size of the CompactChain has decreased compared with Minichain for the m and L values as mentioned earlier. Since the existence and unspent proofs are independent, the proof verification is performed in parallel.

6.4 Transaction witness update

In CompactChain, the existence and unspent proofs change with commitments as new blocks are created in the network. We assume that the number of *inputs* and *outputs* are equal and, Fig. 5 shows that the transaction witness update time for Boneh's, Minichain and CompactChain. In Minichain, the time taken for existence proof update is negligible. Boneh's witness update requires two sequential events for batch deletion of *inputs* from the witness, then batch addition of *outputs* to the witness. In CompactChain, we implemented a witness update similar to the commitments update by exploiting the parallelism. The time taken by a user to update the witness increases linearly with the transaction count in all three protocols.

6.5 Memory consumption

We compare the memory consumption of existing stateless blockchains against Bitcoin. Comparison is analyzed on three aspects (i) RAM Usage, (ii) Disk Usage, and (iii) Memory Pool Usage. The comparison was performed on real-time data of Bitcoin from Jan 2017 to Jan 2022. RAM Usage, considers memory usage while storing stxo-cache in the case of CompactChain and MiniChain, and wheras Bitcoin requires the whole *UTXO set*. Since Boneh has no cache mechanism we consider only current block. Fig. 6 shows the RAM usage from 2017 to 2022. Bitcoin consumes the highest memory, 19 GB due to the ever-growing *UTXO set* size. CompactChain consumes 170.61 MB of RAM. MiniChain consumes 0.05 MB lesser than CompactChain because CompactChain's TXO commitment is 356 bytes larger than MiniChain's. Boneh's consumes the least space due to the lack of caching mechanism, however, underpinned by commitment update speeds. Stateless blockchains can be practically realizable on resource constraint IoT devices as validators due to their low RAM usage. Disk Usage, considers the size required to store the whole blockchain. Fig. 7 shows the disk usage, as expected Bitcoin consumes the highest because one has to store entire blocks of the blockchain. In contrast, stateless blockchains need to store only block headers, therefore

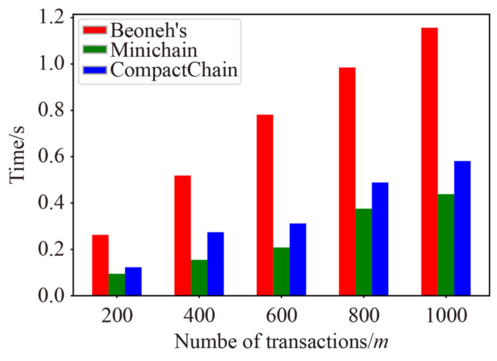

Fig. 5 Performance of the transaction witness update

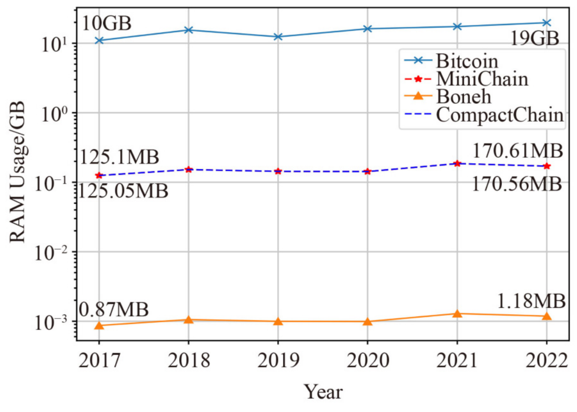

Fig. 6 Comparison of RAM Usage

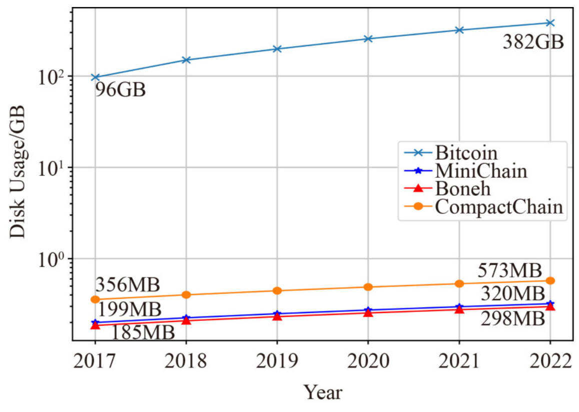

Fig. 7 Comparison of Disk Usage

Table 3 Comparison of Transaction witness size (in bytes) [verification time (s)]

	Existence proof (per Tx)	Unspent proof (per Tx)	Verification time per Tx (parallel execution)	Verification time for 1000 Txs (Parallel execution with 16 threads)
Boneh	–	384 [0.00067]	0.00067 s	0.193 s
Minichain	960 [0.000025]	400 [0.0011]	0.00117 s	0.306 s
CompactChain	384 [0.00056]	400 [0.0010]	0.00115 s	0.303 s

consuming very less space. As pointed out earlier, CompactChain consumes slightly higher header space than MiniChain. Nevertheless, CompactChain consumes 573 MB only which enables edge devices to act as Validators.

Memory Pool or Mempool[3)] Usage considers the size of unconfirmed transactions received and corresponding proofs (if any). A validator in the stateless blockchain need to store the proofs for transactions of the Mempool, whereas Bitcoin does not. Fig. 8 indicates that Bitcoin least space, however, MiniChain consumes enormously high space due to large MMR proofs compared to constant sized Non-Membership witness of CompactChain. Boneh's implementation has only single proof thereby consuming least space among stateless-blockchains.

6.6 Propagation latency

We examine the propagation latency of a block as per the information propagation in the bitcoin network as described in [38]. We have conducted event-driven simulations [39,40,41] for propagation latency. We choose a total of 13000 [42] nodes with 10 miners having hash rate distribution [12] as per in the bitcoin network. We choose the average upload bandwidth of the nodes as 50 Mbps [43].

We assume the average size of a transaction (with single input and single output) as 250 bytes with a total block size of 0.25 MB (1000 transactions). We use the total block validation time of a validator from Fig. 4 (commitments verification time) and Table 3 (transactions verification time). The information to be propagated in the network also consists of the transaction proof with sizes as listed in Table 3. Figure 9 shows the total information propagation from the first observation of a block at the miner to reach all nodes in the network.

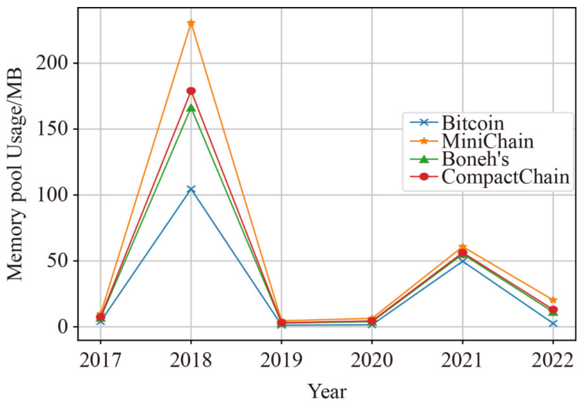

Fig. 8 Comparison of Memory Pool Usage

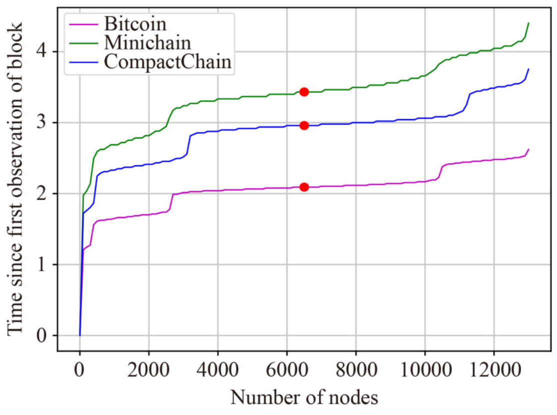

Fig. 9 Performance of propagation latency of a block in the network

In Bitcoin [1], the consensus is defined in probabilistic terms and a block is valid if at least 50 percent of the nodes receive it. So, *consensus latency* is defined as time to reach a block to 50 percent of the nodes in the network. In Fig. 9, the dots in red colour indicate the consensus latency in each protocol.

6.7 Maximum TPS

We test the maximum TPS of the system by considering the performances of transaction verification, commitment updates, and consensus latency. We assume each block with 500 transactions, and each transaction consists of two *inputs* and two *outputs*. Table 4 shows the comparison of maximum TPS in all three frameworks. The results show that the maximum TPS in Boneh's work is restricted to 2.12 TPS due to the complex commitment update. Although having two RSA accumulators CompactChain has improved maximum TPS over Minichain due to a reduction in the size of the information to be propagated in the network.

7 Conclusions and future research

(i) **Summary of research and conclusion** In this paper, we propose a stateless UTXO-model blockchain called compactChain, where the state of the blockchain is comprised of two commitments similar to Minichain. We introduce the RSA accumulator-based TXO commitment, in contrast to the MMR-based TXO commitment in Minichain. We analyze the security of the system based on the strong RSA assumption.

(ii) **Comparison with Boneh's work and Minichain** The theoretical and experimental results show the improvement in the time complexity of the commitment update from $O(m^2)$ to $O(m)$ compare with Boneh's stateless blockchain. The CompactChain reduce the transaction proof size from

Table 4 Comparison of maximum TPS

	Boneh	Minichain	CompactChain
Tx Verification latency (s) (for 500 Txs) [Max TPS]	0.193 [2590.67]	0.306 [1634]	0.303 [1650]
Commitments Update latency (s) [Max TPS]	235.62 [2.12]	0.97 [515.46]	0.99 [505]
Consensus latency (s) [Max TPS]	2.08 [240.38]	3.57 [140]	3.03 [165]
Maximum TPS	2.12	140	165

[3)] Mempool is a database which consists of the valid unconfirmed transactions in the network. A high Mempool size indicates more network traffic which will result in longer average confirmation time and higher priority fees. While creating a new block miner or block producer pick the transactions from Mempool.

$O(log(m)) + O(log(L)) + O(1)$ to $O(1)$ compare with Minichain without compromising the system throughput.

(iii) **Practical implications** CompactChain allows resource constraint devices to become validators due to marginal RAM usage, disk space and less memory pool size, retaining the decentralization property of the blockchain.

(iv) **Paper limitations and future work** The future work will focus on an alternate consensus algorithm to the PoW, which is based on the proof of commitments (PoC) by using the number of transactions as a difficulty level. Sharding technology divides a whole Blockchain network into multiple groups and allows participating nodes to process and store the disjoint set of transactions. For securing the sharded-blockchains, the participating nodes should be reshuffled randomly among the shards, making a node download the state of the newly allocated shard. We focus on integrating the CompactChain with sharded technology to avoid downloading a state for every reshuffle that improves startup time.

Acknowledgements This work was supported by the project Indigenous 5G Test Bed (Building an end to end 5G Test Bed) in India, Dept. of Telecommunication Networks & Technologies (NT) Cell, Government of India. We are extremely grateful for the insightful, helpful and detailed comments of the reviewers. The feedback helps us stay motivated and think more in-depth about the proposed stateless blockchain model and improving this paper.

References

1. Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. See Nakamotoinstitute.org/bitcoin/website, 2008
2. Wood G. Ethereum: a secure decentralised generalized transaction ledger. Ethereum Project Yellow Paper, 2014, 151: 1–32
3. Androulaki E, Barger A, Bortnikov V, Cachin C, Christidis K, De Caro A, Enyeart D, Ferris C, Laventman G, Manevich Y, Muralidharan S, Murthy C, Nguyen B, Sethi M, Singh G, Smith K, Sorniotti A, Stathakopoulou C, Vukolić M, Cocco S W, Yellick J. Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the 13th EuroSys Conference. 2018, 30
4. Dorri A, Kanhere S S, Jurdak R, Gauravaram P. Blockchain for IoT security and privacy: the case study of a smart home. In: Proceedings of 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). 2017, 618−623
5. Song J M, Sung J, Park T. Applications of blockchain to improve supply chain traceability. Procedia Computer Science, 2019, 162: 119–122
6. Azaria A, Ekblaw A, Vieira T, Lippman A. MedRec: using blockchain for medical data access and permission management. In: Proceedings of the 2nd International Conference on Open and Big Data (OBD). 2016, 25−30
7. Pincheira M, Vecchio M, Giaffreda R, Kanhere S S. Cost-effective IoT devices as trustworthy data sources for a blockchain-based water management system in precision agriculture. Computers and Electronics in Agriculture, 2021, 180: 105889
8. Pee S J, Kang E S, Song J G, Jang J W. Blockchain based smart energy trading platform using smart contract. In: Proceedings of 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC). 2019, 322−325
9. Bitcoin. Full node. See Bitcoin.org/en/full-node website, 2023
10. Delgado-Segura S, Pérez-Solà C, Navarro-Arribas G, Herrera-Joancomartí J. Analysis of the bitcoin UTXO set. Financial Cryptography and Data Security. 2019, 78-91
11. Bitcoin. Bitcoin repository. See GitHub.com/bitcoin/bitcoin website, 2023
12. Blockchain. Block-size and UTXO charts. See Blockchain.com/explorer/charts website, 2023
13. Barić N, Pfitzmann B. Collision-free accumulators and fail-stop signature schemes without trees. In: Proceedings of International Conference on the Theory and Application of Cryptographic Techniques Konstanz on Advances in Cryptology. 1997, 480−494
14. Camenisch J, Lysyanskaya A. Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology. 2002, 61−76
15. Li J, Li N, Xue R. Universal accumulators with efficient nonmembership proofs. In: Proceedings of the 5th International Conference on Applied Cryptography and Network Security. 2007, 253−269
16. Boneh D, Bünz B, Fisch B. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: Proceedings of the 39th Annual International Cryptology Conference on Advances in Cryptology. 2019, 561−586
17. Tremel E. Real-world performance of cryptographic accumulators. See cs.brown.edu/research/pubs/theses/ugrad/2013/tremel website, 2013
18. Chen H, Wang Y MiniChain: a lightweight protocol to combat the UTXO growth in public blockchain. Journal of Parallel and Distributed Computing, 2020, 143: 67−76
19. Todd P. Making UTXO set growth irrelevant with low-latency delayed TXO commitments. See Petertodd.org/2016/delayed-txo-commitments website, 2016 Shamir, A. How to share a secret. Communications of the ACM, 1979, 22(11): 612–613
20. Perard D, Lacan J, Bachy Y, Detchart J. Erasure code-based low storage blockchain node. In: Proceedings of 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). 2018, 1622−1627
21. Kadhe S, Chung J, Ramchandran K. SeF: a secure fountain architecture for slashing storage costs in blockchains. 2019, arXiv preprint arXiv: 1906.12140
22. Raman R K, Varshney L R. Dynamic distributed storage for scaling blockchains. 2017, arXiv preprint arXiv: 1711.07617
23. Shamir, A. How to share a secret. Communications of the ACM, 1979, 22(11): 612−613
24. Rashmi K V, Shah N B, Kumar P V, Ramchandran K. Explicit construction of optimal exact regenerating codes for distributed storage. In: Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing. 2009, 1243−1249
25. Rawat A S, Koyluoglu O O, Silberstein N, Vishwanath S Optimal locally repairable and secure codes for distributed storage systems. IEEE Transactions on Information Theory, 2014, 60(1): 212−236
26. Matzutt R, Kalde B, Pennekamp J, Drichel A, Henze M, Wehrle K. How to securely prune bitcoin's blockchain. In: Proceedings of 2020 IFIP Networking Conference (Networking). 2020, 298−306
27. Chepurnoy A, Larangeira M, Ojiganov A. Rollerchain, a blockchain with safely pruneable full blocks. 2016, arXiv preprint arXiv: 1603.07926
28. Buterin V. The stateless client concept. See Ethresear.ch/t/the-stateless-client-concept/172 website, 2017
29. Chepurnoy A, Papamanthou C, Srinivasan S, Zhang Y. Edrax: a cryptocurrency with stateless transaction validation. IACR Cryptol. ePrint Arch. 2018/968. 2018
30. Dahlberg R, Pulls T, Peeters R. Efficient sparse Merkle trees: caching strategies and secure (non-)membership proofs. In: Proceedings of the

21st Nordic Workshop on Secure Computer Systems, 2016
31. Menezes A J, Rosen K H, van Oorschot P C, Vanstone S A. Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997
32. Lipmaa H. Secure accumulators from Euclidean rings without trusted setup. In: Proceedings of the 10th International Conference on Applied Cryptography and Network Security. 2012, 224−240
33. Wesolowski B. Efficient verifiable delay functions. In: Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology. 2019, 379−407
34. Sun D Z, Cao Z F, Sun Y. How to compute modular exponentiation with large operators based on the right-to-left binary algorithm. Applied Mathematics and Computation, 2006, 176 (1): 280−292
35. Fiat A, Shamir A. How to prove yourself: practical solutions to identification and signature problems. In: Proceedings of Advances in Cryptology. 1987, 186−194
36. Etremel. Crypto-accumulators. See Github.com/etremel/crypto-accumulators/blob/master/lib/flint/BigInt website, 2013
37. Reddy B S, Reddy T U K. See Github.com/TUdayKiranReddy/Compact-Chain website, 2023
38. Decker C, Wattenhofer R. Information propagation in the bitcoin network. In: Proceedings of IEEE P2P 2013 Proceedings. 2013, 1−10
39. Reddy B S, Sharma G V V. Optimal transaction throughput in proof-of-work based blockchain networks. Proceedings, 2019, 28(1): 6
40. Reddy B S, Sharma G V V. UL-blockDAG: unsupervised learning based consensus protocol for blockchain. In: Proceedings of the 40th IEEE International Conference on Distributed Computing Systems (ICDCS). 2020, 1243−1248
41. Reddy B S, Sharma G V V. Scalable consensus protocols for PoW based blockchain and blockDAG. 2020, arXiv preprint arXiv: 2010.05447
42. Bitnodes. See Bitnodes.io website, 2023
43. Speedtest. Global-index. See Speedtest.net/global-index website, 2021

B Swaroopa Reddy is currently pursuing the PhD degree in the Department of Electrical Engineering, Indian Institute of Technology Hyderabad, India from 2017. He received the BTech degree in Electronics and Communication Engineering from Sri Krishnadevaraya University, India in 2009. From 2010 to 2017, he was with the Bharath Sanchar Nigam Limited (BSNL), a Public Sector Unit under the Government of India. His research interests include scalability of Blockchain Networks, Privacy and Security in Blockchain, ZK-Rollups, Decentralized Identities (DiD) and Verifiable Credentials (VC).

T Uday Kiran Reddy is currently pursuing the BTech degree in the Department of Electrical Engineering, Indian Institute of Technology Hyderabad, India from 2019. His current research interests include the areas of Stochastic Optimisation, Signal Processing, Machine Learning and Blockchain.