Efficient Verifiable Boolean Range Query for Light Clients on Blockchain Database

Jianpeng Gong, Jiaojiao Wu, Jianfeng Wang, and Shichong Tan(✉)

State Key Laboratory of Integrated Service Networks (ISN), Xidian University, Xi'an 710071, China
{jpgong,jiaojiaowujj}@stu.xidian.edu.cn, jfwang@xidian.edu.cn, sctan@mail.xidian.edu.cn

Abstract. Blockchain allows clients to query and verify any transactions, which requires the clients to maintain the entire blockchain database locally. This approach is inadvisable because the blockchain database is an append-only ledger and incurs significant maintenance overhead. Very recently, blockchain light client has attracted considerable concerns, which relies on a third party (i.e., a full node) to perform query processing and verification. However, the dishonest full node may return an incorrect and incomplete result of the query requests. Therefore, it remains a challenging issue to achieve secure, efficient, and rich verifiable queries for light clients. In this paper, we propose an efficient verifiable Boolean range query scheme for light clients on the blockchain database. Firstly, we design a new authenticated data structure, polynomial commitment B^+-tree (PCB-tree), which efficiently ensures the correctness and completeness of Boolean range queries for blockchain light clients. Secondly, we provide a tunable trade-off between query time and communication overhead by autonomously setting the fanout size of the PCB-tree. Moreover, our scheme can support batch processing to reduce query complexity and proof size. Finally, security analysis and performance evaluation show that our proposed scheme is secure and practical.

Keywords: Blockchain database · Light clients · Verifiable boolean range query · Data integrity

1 Introduction

Blockchain, as a revolutionary technology [10], has aroused widespread attention and research in various fields, such as smart contract platform, decentralized storage, and supply chain traceability. Meanwhile, with the popularization of blockchain technology in the finance and supply chain, the people's demand for efficient and various queries of data stored in a blockchain database has become more and more urgent. For illustration, a user, Bob, wants to query the data about his consumption in the last month that satisfy the following Boolean range conditions, such as "[2021-11-01, 2021-12-01]" and "$sender$ = Bob ∨ $receiver$

J. Zhou et al. (Eds.): ACNS 2022 Workshops, LNCS 13285, pp. 15–35, 2022.
https://doi.org/10.1007/978-3-031-16815-4_2

= Bob", where $\vee$ represents Boolean logical operator OR. The result will be faithfully returned if the query is conducted in the traditional centralized systems with a trusted party. From the security perspective, if the client downloads the complete blockchain duplication as a full node, it can query and validate the integrity of transactions locally. However, The appended-only and immutable properties of blockchains result in the data increases with the generation of new blocks, which requires a large amount of storage and network overhead [7]. In the past two years, the data on the Ethereum blockchain has been growing linearly with a slope of roughly 0.424 GB/Day is significantly faster than Bitcoin, which exceeds the capability of most query clients. To address above concern, most blockchain systems introduce the light client (*e.g.*, Simplified Payment Verification [10] and Light Ethereum Subprotocol [18]), which can download only the valid block header from the longest chain to verify whether the current block contains the interested transactions without the complete blockchain dataset.

However, the light client that stores only block header information will raise the following concerns when querying transactions:

- **The integrity of query result:** The light client relies on the query service provided by the full node. If the full node is untrusted, it probably returns fake or partial results to light clients [3], or it will obtain the privacy information of light clients by capturing some sensitive request [8,13]. Therefore, the security of current blockchain queries is still a crucial issue.
- **The query efficiency:** The increasing growth of blockchain brings heavy overhead for developers to access transactions on the blockchain. Meanwhile, the current blockchain system only supports single-type queries and has low query efficiency. Therefore, it is necessary to implement a blockchain query system with high efficiency and rich query functionalities for light clients.

There are some attempts to implement verifiable queries of blockchain for the light client. The state-of-the-art schemes either utilize the trusted execution environment or authenticated data structure to ensure the integrity of query results. Therefore, it is of great significance to design a new structure that guarantees the correctness and completeness of the query results and reduces the communication and verification costs.

1.1 Contributions

In this paper, we focus on verifiable Boolean and range queries. Motivated by the above observations, we propose an efficient and verifiable Boolean range query scheme for light clients on the blockchain database. To evaluate our design, we implement the prototype system and conducted multiple experiments based on it. Our main contributions can be summarized as follows:

- We propose a new authenticated data structure, polynomial commitment B^+-tree (PCB-tree), that supports the Boolean and range queries.
- We provide an adjustable trade-off between query time and communication overhead by autonomously setting the fanout size of the PCB-tree. Meanwhile, our scheme can support batch processing to reduce the proof size and verification time.

– We prove the security of the scheme in theory and implement a prototype system to evaluate the performance of our proposed scheme. The experiment results demonstrate that our scheme is efficient in terms of query, verification and communication overheads.

1.2 Related Work

In this section, we briefly review the related works on verifiable query processing over traditional outsourced databases and blockchain.

Verifiable Query Processing over Traditional Database. There are some verifiable query works that have been studied in outsourced databases. The current verifiable query is divided into two categories: circuit-based Verifiable Computation (VC) technique and Authenticated Data Structure (ADS). However, the VC-based scheme overhead is very high and sometimes impractical [11]. In comparison, the ADS-based approach is generally more efficient. The Merkle Hash Tree (MHT) is a significant component in verifiable query schemes and is extended to different types of databases, including Merkle B-tree (MB-tree) for relational data [6] and Merkle R-tree (MR-tree) for spatial data [20]. However, these works are more for the outsourced databases and insufficient for the blockchain case.

Verifiable Query Processing over Blockchain. Simplified Payment Verification (SPV) protocol[1] is the first light client protocol proposed in the Bitcoin paper [10]. It can use Merkle proof to verify whether the blockchain network accepts a transaction. However, the costs of verification and storage increase linearly with the growth of the blockchain. Some verifiable query works have been studied in blockchain systems to ensure the integrity of the query results. Xu et al. [19] present an accumulator-based ADS and implement a verifiable query framework, called vChain, that alleviates the storage and computing costs of the light client. However, the public key size of the accumulator is linear to the largest multiset size, and the large proof size leads to an expensive communication overhead for the light client. Shao et al. [16] utilize the Trusted Execution Environment (TEE) to achieve an authentication range query scheme for blockchain, but it does not discuss potential side-channel attacks against TEE. Zhu et al. [23] put forward a verifiable aggregate queries scheme based on the accumulator that supports multiple selection predicates. However, it uses the same accumulator as vChain and does not solve the linear overhead problem. Meanwhile, the construction cost of ADS is expensive since different query dimensions require different ADSs. Zhang et al. [21,22] utilize MB-tree structure and cryptographic accumulator to present a gas-efficient scheme for hybrid-storage blockchains. However, its index maintenance cost remains relatively expensive, and the query process is complicated. LineageChain [15] leverages a novel skip list index to achieve efficient provenance query processing and stores provenance information

[1] The concepts of payment verification and transaction verification are different in the blockchain.

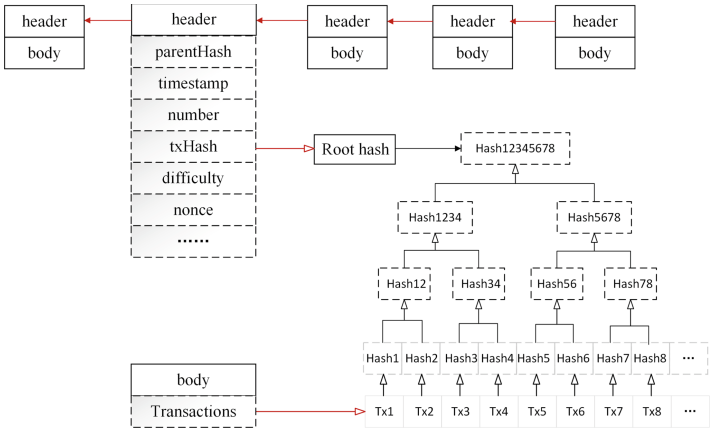


Fig. 1. Blockchain data structure

in a Merkle tree. Therefore, LineageChain only ensures the correctness of the query results, but not completeness.

2 Preliminaries

2.1 Blockchain Data Structure

From the perspective of data structure, blocks in blockchain mainly contain two parts: block header and block body. As shown in Fig. 1, all transactions are recorded in each block and organized a MHT built on top of them. The block header contains: (1) *parentHash*, which is the hash of the previous block; (2) *timestamp*, which is the time of block creation; (3) *number*, which is block height; (4) *txHash*, which is the root hash of MHT; (5) *difficulty*, which is the difficulty coefficient of mined blocks; (6) *nonce*, which is the random number constructed by the miners to solve Proof of Work (PoW) protocol problem. Other miners can append it to the blockchain after verifying the *nonce* of the new block.

2.2 B^+-Tree

B^+-tree [4] is a multi-branch sort tree structure that can improve the search efficiency of range queries. The numerical range query process of B^+-tree is described as follows:

- One starts the query from the root node. If the lower bound of the range query matches the current non-leaf node, it then searches its subtree.
- When traversing to a leaf node, one adds the corresponding record into the results set if the lower bound element is found, then continue to search backwards through the pointer relationship between nodes until the upper bound is found at the end of the query.

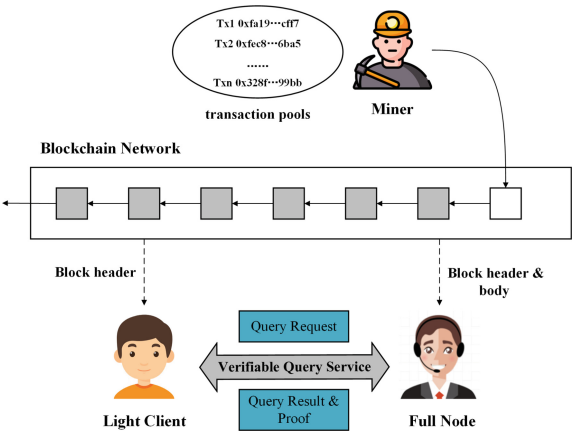


Fig. 2. Our system architecture

2.3 Constant Size Polynomial Commitment

A polynomial commitment scheme allows one to commit a polynomial with a short proof while keeping it hidden from others. The verifier can confirm the claimed statement of the committed polynomial. Kate, Zaverucha and Goldberg [5] first present polynomial commitment scheme (KZG commitments) as the following:

- **Setup**$(1^\lambda, t)$ generates an appropriate algebraic structure $\mathcal{G}$ and a public-private key pair $\langle \mathsf{PK}, \mathsf{SK} \rangle$ to commit to a polynomial of degree $\leq t$.
- **Commit**$(\mathsf{PK}, p(x))$ generates a commitment $\mathcal{C}$ to a polynomial $p(x)$ using the public key PK.
- **CreateWitness**$(\mathsf{PK}, p(x), z)$ generates a witness ω for the evaluation $p(z)$ of $p(x)$ at the index z.
- **VerifyEval**$(\mathsf{PK}, \mathcal{C}, z, p(z), \omega)$ verifies that $p(z)$ is indeed the evaluation at the index z of the polynomial committed in $\mathcal{C}$.
- **CreateWitnessBatch**$(\mathsf{PK}, p(x), S)$ generates the batched witness ω_S for the value $p(i)$, where $i \in S$.
- **VerifyEvalBatch**$(\mathsf{PK}, \mathcal{C}, S, r(x), \omega_S)$ verifies the correctness of the witness returned by **CreateWitnessBatch** algorithm.

3 Problem Statement

In this section, we describe the system model and threat model of our scheme.

3.1 System Model

We propose a verifiable query scheme on blockchain database. Figure 2 depicts the four entities in our system framework:

Table 1. Data formal definition

tx_{id}	t_i	V_i	W_i
tx_1	2021-05-12	5	{addr1, addr2}
tx_2	2021-05-21	35	{addr3, addr2}
tx_3	2021-04-28	20	{addr5, addr1}
tx_4	2021-05-21	80	{addr4, addr1}
tx_5	2021-05-22	56	{addr6, addr8}
tx_6	2021-05-19	90	{addr3, addr1}
tx_7	2021-04-22	10	{addr3, addr1}

- **Blockchain Network:** A network of untrusted nodes collectively maintains the blockchain data and guarantees stored data is immutable. We assume our system is based on an account-based blockchain.
- **Full Node:** A full node downloads complete duplication of the blockchain database and can independently verify the correctness of any block/transaction [1]. Also, the full node can provide payment services for others, such as query or Application Programming Interface (API) services.
- **Light Client:** A light client only stores block headers and verifies transactions relying on full nodes. It generally runs on resource-constrained devices.
- **Miner:** A miner[2] competes to create new blocks by a consensus algorithm (*e.g.*, PoW algorithm) and appends it to the blockchain network.

In our system, when a light client wants to retrieve existing transaction data, the light client firstly synchronizes all newer block headers from the longest chain and connects to one of the full node servers to send query requests in which the client is interested. The miners are responsible for organizing all transactions within the block to construct the PCB-tree and appending root commitment to the block header to replace the traditional root hash. The full node provides rich queries for light clients and returns both query results and the corresponding proofs using our ADS structure. After that, the clients use the Verification Object (VO) to verify the correctness and completeness of returned results.

As shown in Table 1, the transaction data tx_i is defined as triple elements $\langle t_i, V_i, W_i \rangle$, where t_i is the transaction timestamp, V_i is a transaction value that represents one numerical attribute, and W_i is a set attribute that contains the address information of the sender and receiver. Each block contains multiple transaction data objects $\{tx_1, tx_2, \cdots, tx_n\}$. Light clients want to query all transactions that match the query request in a period. In this paper, we consider mainly the rich Boolean range queries based on the time window. Specifically, a Boolean range query is defined as follows: $q = \langle [t_s, t_e], [v_l, v_u], \gamma \rangle$, where t_s and t_e is the start and end time of a time window, v_l and v_u represent the lower and upper bound of the transaction query range, and γ is a Boolean query such as addr1$\wedge$(addr2$\vee$addr3). For example, the light client may request a specific query $q = \langle [2021-05, 2021-06], [10, 40], sender = \text{addr1} \wedge receiver = \text{addr2} \rangle$ to find all the matched transactions.

[2] Miners can be full nodes or light nodes.

3.2 Threat Model and Assumptions

In our scheme, we assume the blockchain network is strong and does not consider some attacks against blockchain, *e.g.*, Eclipse attack, and Sybil attack. We believe that light clients are honestly reliable and randomly connect to one of the full nodes from the blockchain network to perform the query operation. Furthermore, driven by economic interests, we assume the miners are honest to faithfully execute our ADS structure, which will not lead to some underlying system security vulnerabilities. Meanwhile, we think no particular relationship between miners and full nodes. The full nodes are untrusted and regarded as potential adversaries. On the one hand, the full nodes can obtain some sensitive information (*e.g.*, account address and transaction information) by the queries of clients, which will result in the disclosure of user privacy [3,8]. On the other hand, the full nodes may return incorrect or incomplete query results to reduce the query expense [19]. Hence, the query results from the full nodes need to be validated to satisfy the following criteria:

- **Correctness.** The result data tuples indeed exist in the blockchain databases, and they have not been tampered with in any way. Meanwhile, as the results, they should satisfy the query conditions.
- **Completeness.** No satisfactory results have been omitted by the full nodes, either intentionally or unintentionally.
- **Lightweightness.** The results should have lower storage costs and communication overhead for lightweight clients than that of current schemes.

4 Polynomial Commitment B$^+$-Tree

4.1 Overview

The MHT is usually constructed for each block to authenticate transaction data in the original blockchain. However, this naive method has the following shortcomings. Firstly, the MHT supports only efficient membership queries instead of providing non-membership proofs. Secondly, the proofs of MHT can hardly be aggregated effectively, resulting in serious communication overhead and inefficient verification. To deal with the drawbacks above, we propose a novel ADS structure illuminated by [6] and [17], PCB-tree, which ensures the integrity of light client queries and avoids the problem of large public key parameters and proofs.

4.2 PCB-Tree Structure on Blockchain

For simplicity, we combine the polynomial commitment and B$^+$-tree to implement a PCB-tree supporting constant size intra-node proof and more efficient queries than MHT. Denote (k_i, tx_i) the key-value pair of PCB-tree leaf node, where k_i represents the numerical attribute (*e.g.*, transferred transaction's amount), tx_i is the corresponding transaction data. In the B$^+$-tree index, the

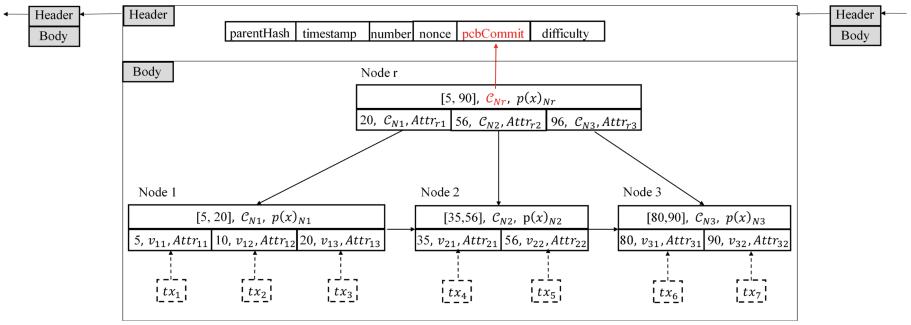


Fig. 3. The ADS structure on a block

overflow page is commonly used to deal with duplicate keys [14]. For the convenience of description, we assume there are no duplicate keys in the context.

Figure 3 shows a block structure with PCB-tree. The block header consists of the following elements: *parentHash*, *timestamp*, *number*, *pcbCommit*, *difficulty*, *nonce*, where *pcbCommit* is the root commitment of PCB-tree to replace the original *txHash* (Fig. 1). In the PCB-tree, each tree node has four fields: the minimum and maximum key (denoted by $[min, max]$), the node commitment value (denoted by $\mathcal{C}$), the polynomial (denoted by $p(x)$), and the transactions set of each node (denoted by S). Let $LagrangeInterpolation(\cdot)$ be a function to find a polynomial, '$\|$' be the string concatenation, $hash(\cdot)$ be Keccak-256 algorithm, respectively. The fields of a node are described as follows:

- $S = [(k_1, v_1), \ldots, (k_b, v_b)]$, where b denotes the number of key-value pairs in each node, k_i is the index. For the leaf node, v_i represents a transaction hash such that $v_i = hash(tx_i)$. For the non-leaf node, v_i represents a commitment value $\mathcal{C}_i$ of its child node.
- $k'_i = hash(k_i \parallel i)$, $S' = [(k'_1, v_1), \ldots, (k'_b, v_b)]$. To account for the sorted order of the keys and the position binding relation within the nodes, we transform the k_i to k'_i.
- $p(x)$ represents a polynomial for key-value pairs in each node such that $p(k'_i) = v_i$. It can be done with the Lagrange interpolation formula or Horner's method.
- $\mathcal{C} = \textbf{Commit}(\mathsf{PK}, p(x))$, where PK is the public key generated in $\textbf{Setup}(\cdot)$ algorithm in advance.
- *Attr* represents attributes set for each node. For the leaf node, it is a set of transaction attributes, including the boundary keys information, the address information and so on. For the non-leaf node, $Attr_n = Attr_{l_1} \cup \ldots \cup Attr_{l_b}$, where $l_1, \ldots, l_b$ are the children of node n.

Algorithm 1 describes the ADS construction procedure, and the PCB-tree is built based on the transaction objects of the block in a bottom-up fashion. First, the miner parses each transaction tx_i in transaction set T into the formal we

Algorithm 1: ADS Construction (by the miner)

Input: Public key PK, Transactions set T
Output: root commitment *pcbCommit*

Initialize a new PCB-tree;
Parse T into key-value pairs $List_{tx} \leftarrow \{(k_1, tx_1, attr_1), \cdots, (k_n, tx_n, attr_n)\}$;
Sort $List_{tx}$ according to the k_i;
Number of leaf node $l \leftarrow n/f$;
for *i=1 to l* **do** `// leaf node`
  // entries of each node;
  for *j=1 to f* **do**
    $k_j \leftarrow k_{i+j}$;
    $v_j \leftarrow hash(tx_{i+j})$;
    $Attr_j \leftarrow attr_{i+j}$;
  end
  $S_i \leftarrow [(k_1, v_1), \cdots, (k_f, v_f)]$;
  $Attr_i \leftarrow [Attr_1, \cdots, Attr_f]$;
  $\mathcal{C}_i, p(x)_i \leftarrow$ **nodeUpdate**$(S_i, node_i)$;
  Insert $\langle \mathcal{C}_i, p(x)_i, S_i, Attr_i \rangle$ to i-th PCB-tree node;
end
repeat
  for *i=1 to l/f* **do** `// non-leaf node`
    for *j=1 to f* **do**
      $k_j \leftarrow k_{i+j}$;
      $v_j \leftarrow C_l$;
      $Attr_j \leftarrow Attr_l$;
    end
    $S_i \leftarrow [(k_1, v_1), \cdots, (k_f, v_f)]$;
    $Attr_i \leftarrow [Attr_1, \cdots, Attr_f]$;
    $\mathcal{C}_i, p(x)_i \leftarrow$ **nodeUpdate**$(S_i, node_i)$;
    Insert $\langle \mathcal{C}_i, p(x)_i, S_i, Attr_i \rangle$ to i-th PCB-tree node;
  end
  $l \leftarrow l/f$;
until *all transactions is completed*;
Store root commitment *pcbCommit* in block header;

defined. Next, the miner inserts each transaction data to the leaf node of PCB-tree and updates the commitment value $\mathcal{C}$ and polynomial $p(x)$ for each node. This process is repeated until all transactions are inserted. Finally, after the tree construction is finished, the root commitment of the PCB-tree will be written in the block header as *pcbCommit*. The node update algorithm **nodeUpdate**() aims to update the value of commitment and polynomial for inserted node. We first build the binding relationship between the numerical attribute k_i and position i, then calculate the polynomial by the Lagrange interpolation formula for the node and its corresponding commitment value. The procedure is described in Algorithm 2.

Algorithm 2: PCB-tree Node Update Algorithm

Function `nodeUpdate`$(S,$ *curr*`):`
  Input: The key-value pairs of node S, The updated node *curr*
  Output: The commitment $\mathcal{C}$, The polynomial $p(x)$
  for *i=1 to S.size* **do**
    $k'_i \leftarrow hash(k_i \parallel i)$;
  end
  $S' \leftarrow [(k'_1, v_1), \cdots, (k'_{S.size}, v_{S.size})]$;
  $p(x) \leftarrow LagrangeInterpolation(S')$;
  $\mathcal{C} \leftarrow$ **Commit**$(PK, p(x))$;
  return $\mathcal{C}, p(x)$;

Our PCB-tree leverages the feature of B^+-tree to improve the efficiency of range queries and reduce the I/O operation times of node queries. The query efficiency of our PCB-tree is higher than MHT. In this paper, the size of public key PK grows linearly with the branching factor f of the PCB-tree, rather than the largest transaction set size. It can be adjusted flexibly to make the trade-off between query efficiency and communication overhead. Meanwhile, the public-private key pair of KZG commitments can be generated by executing the Distributed Key Generation (DKG) protocol [12], and the PK can be shared to a bulletin board which can be generated once and then reused.

5 The Proposed Construction

For ease of illustration, we first focus on the range query in each block. We then extend it to the Boolean query and ensure the integrity of its results. Finally, to enhance the performance of the query service, we discuss batch processing on multiple query objects. Based on our designed PCB-tree, we explain the proof generation and verification for the range query.

5.1 Verifiable Range Query Processing

In the verifiable query phase, when the light client triggers a query request, the full node parses the query firstly and returns the correct results R and proofs VO according to Algorithm 3. Then, the light client updates the newest block headers periodically and verifies the integrity of the results through VO. Next, we will focus on the range query processing on numerical attribute V_i for the full node.

In the full node, the range query is executed in a top-down way that is similar to the range query of the B^+-tree. Algorithm 3 shows a range query $q = [l, u]$ on a single block. When l equals u, the range query is a point query. First, the full node can process a query from the root node. If the query condition does not intersect with the attribute of the current node, it means its subtree does not contribute to the query result. In this case, the full node will generate the proof for the root node as the VO, and the procedure is terminated. Otherwise, keep exploring its subtree. During the search process, if the keyword on the non-leaf node is equal to the given value, it only adds the corresponding proof to VO and does not terminate until the real data in leaf nodes are found. The leaf node of the PCB-tree organizes a sequence list, and we can traverse backward from the first leaf node found. The query algorithm of PCB-tree **RangeQuery**() is described in Algorithm 5 (See Appendix A), which recursively queries each level of nodes. To improve the query efficiency of internal nodes, we use an efficient localization algorithm **getPos**(). It mainly utilizes the idea of binary search to locate the query path quickly, and we use the bit vector to denote which keys are the points on the search path in each node. For example, the $\langle 0, 1, 1, 0 \rangle$ means that the second and third positions of the current node are retrieved. Algorithm 6 shows the localization procedure in detail (See Appendix A).

Algorithm 3: Range Query Processing (by full node)

Input: PCB-tree *root*, Range query condition $q = [l, u]$
Output: Query Results R, Verification Object VO

Initialize two empty set R, VO;
if *root.[min,max] matches q* **then**
  RangeQuery($root$, l, u, R, VO);
else
  $R = \emptyset$;
  $num \leftarrow root$.keyNum, $p(x) \leftarrow root.p(x)$;
  ω_{min} = **CreateWitness**(PK, $p(x)$, k_0);
  add $\langle 0, (k_0, v_0), \omega_{min}\rangle$ to VO;
  ω_{max} = **CreateWitness**(PK, $p(x)$, k_{num});
  add $\langle num, (k_{num}, v_{num}), \omega_{max}\rangle$ to VO;
end
return $\langle R, VO\rangle$

For example, consider a range query $q = [19, 40]$ as shown in Fig. 3. The full node traverses the keyword index from the root node to the leaf nodes, and adds the matched transaction to R. Finally, the results are $\{[(3, 20, tx_3), (1, 35, tx_4)]\}$. In this case, the full node needs to return the membership of the query results and the non-membership of the greatest and smallest elements in the range, *e.g.*, 19,40. The VO returned by the full node includes $\{[\langle 1, (20, \mathcal{C}_{N1}), \omega_{r1}\rangle, \langle 2, (56, \mathcal{C}_{N2}), \omega_{r2}\rangle], [[\langle 2, (10, v_{12}), \omega_{12}\rangle, \langle 3, (20, v_{13}), \omega_{13}\rangle], [\langle 1, (35, v_{21}), \omega_{21}\rangle, \langle 2, (56, v_{22}), \omega_{22}\rangle]]\}$, where ω_{ij} is the witness for the elements in the j-th position of the i-th node and is generated by invoking **CreateWitness**$(\cdot)$ algorithm.

Algorithm 4 describes the steps of result verification on the light client. The verification process is on the client-side from top to bottom. At first, the light client downloads all block headers from the blockchain to fetch the root commitment against the *pcbCommit* and leverages the polynomial commitment primitive **VerifyEval**$(\cdot)$ algorithm to check the correctness of the search path elements. The binding relationship between data and position in each node ensures the completeness of query results. The validated commitment for each entry in the parent node needs to be used to verify the correctness of their child nodes. In the leaf node, the user uses the function $hash(\cdot)$ to compute $v_i = hash(tx_i)$ according to R, then invoke **VerifyEval**$(\cdot)$ algorithm to prove the correctness of returned result.

5.2 Extension to Verifiable Boolean Query

The previous section mainly discusses the range queries on the numerical attribute V_i. In real-life scenarios, the query client may consider the keyword queries on the set attributes W_i. The Boolean query on the set attributes is supported in our PCB-tree by the field *Attr*. In the non-leaf nodes, the attribute set of the parent node is the union of the attribute sets of all its child nodes. Therefore, when the full node receives a Boolean query condition $q = \{addr1 \wedge (addr3 \vee addr4)\}$, it firstly parses the query into two parts: $\{addr1\}$

Algorithm 4: Results Verification (by the light client)

Input: Query results R, Verification object VO, Query condition q
Output: The verification result: 1 or 0

Interpret R and VO as a list of $\langle i, k_i, tx_i \rangle$ and $\langle i, (k_i, v_i), \omega_i \rangle$, respectively;
$vCommit \leftarrow$ root commitment $PcbCommit$;
for *each level in VO* **do**
  $vo \leftarrow \langle i, (k_i, v_i), \omega_i \rangle$;
  Check the k_i is in the query range q;
  Verify the i-th entry of current node is correct via the $vCommit$ and vo;
  $vCommit \leftarrow v_i$;
  if *current node is leaf and* k_i *matches* q **then**
    $value \leftarrow hash(tx_i)$ according to the R;
    Check the $value$ is equal the v_i of the VO;
  end
end

and $\{addr3, addr4\}$. In this paper, the Boolean query is represented by a Boolean function in Conjunctive Normal Form (CNF), which is a list of AND or OR operators. The full node starts the query from the root node and compare query condition one by one with the set attribute *Attr* in each node. However, this way is not efficient. In order to speed up the query, we introduce Bloom Filter (BF) into PCB-tree. Bloom filter is a long binary vector and a series of random mapping functions, and it can be used to test whether an element is a member of a set fastly. When constructing the index of PCB-tree, we need to create a BF bit vector for each node attribute, which means each BF represents a set attribute W_i, and the BF of non-leaf nodes denote the union of BFs in its child nodes. Therefore, when the system starts traversing from the PCB-tree root node, BF is used to determine whether the subtrees of current node have the query attributes.

In the range query, v_i is the hash of transactions tx_i, and it is seen as an authenticator of the transactions. Based on polynomial commitment, we guarantee the integrity of numerical attribute query results. In order to guarantee the integrity of the Boolean query, we need to build a binding relationship between numerical attributes, set attributes and transactions. Hence, v_i needs to be transformed into a tamper-proof digest value, such as hash value, $v_i = hash(tx_i \parallel W_i)$. We take v_i as the value in the key-value pair (k_i, v_i) in the leaf node for Boolean range queries.

Remark 1. In order to further reduce the communication overhead, it is necessary for supporting batch operations. The primitive **CreateWitnessBatch**$(\cdot)$ introduced in Sect. 2.3, which can aggregate multiple query points in the same polynomial commitment. We can aggregate the proof for multiple query points under the same node when executing range query. For example, the full node can return an aggregated proof $\omega_{r1,r2} =$ **CreateWitnessBatch**$(\mathsf{PK}, p(x)_{Nr}, [(20, \mathcal{C}_{N1}), (56, \mathcal{C}_{N2})])$ for two points under the root node $\langle 1, (20, \mathcal{C}_{N1}), \omega_{r1} \rangle$ and $\langle 2, (56, \mathcal{C}_{N2}), \omega_{r2} \rangle$. The light client can apply **VerifyEvalBatch**$(\cdot)$ to process

batch verification. Boneh et al. [2] proposes two polynomial commitment schemes which can open proof for multiple points and polynomials at the same time. We can also leverage this enhanced polynomial commitment scheme to aggregate different node in the PCB-tree which will further reduce the VO size.

5.3 Security Analysis

In this paper, we give a formal definition and analyze the security of our proposed scheme. Note that the polynomial commitment scheme is secure [5].

Definition 1 *(Security). A verifiable Boolean range query scheme is secure if the success probability of any polynomial-time adversaries in the following experiment is negligible:*

- *Run the ADS generation algorithm and send all transactions* $\{tx_1, \ldots, tx_n\}$ *in a block to the adversary;*
- *The adversary outputs the query* q*, the result* R*, and the VO.*

The above definition indicates that malicious full node forges an incorrect or incomplete result is negligible. Next, we will prove that our proposed scheme indeed satisfies the desired security requirements.

Theorem 1. *Our proposed verifiable Boolean range query scheme based on PCB-tree can guarantee the correctness and completeness of query result as defined in Definition 1.*

Proof. The verifiable query processing should guarantee that the returned results are correct and complete. We prove this theorem by contradiction as follows:

(1) *Correctness of query results.* The returned results R contain a transaction tx^* such that $tx^* \notin \{tx_i\}_{i=1}^n$ and pass the verification. The client will validate the integrity of the transaction with respect to the *PcbCommit* stored in the blockchain. Therefore, the forge is impossible because the polynomial commitment scheme and the underlying consensus mechanism of the blockchain are secure.

(2) *Completeness of query results.* There exists a transaction tx_d that satisfies the query condition q, but not in the result set R. Now suppose there is a missing transaction tx_d. In our proposed PCB-tree, all transactions are stored in the leaf nodes after being sorted according to the numerical attribute k_i, and we build the binding relationship between k_i and the indexed position i for commitment. During query processing, the full node requires two additional boundary objects for non-membership proofs for query objects, falling immediately to the left and the right of tx_d. Meanwhile, the light verifier syncs the latest block headers from the blockchain network. The missing object tx_d must fall under one polynomial witness value in the VO. Thus, our proposed scheme is secure.

6 Performance Evaluation

This section describes the performance evaluation of our verifiable query scheme. We deploy all experiments on a personal laptop computer with AMD R7 4800H CPU @ 2.90 GHz, 24 GB RAM, and run a single thread to simulate the processing of the full node and the light client. In the experiments, we retrieve the Ethereum databases via a blockchain infrastructure, *e.g.*, Infura[3]. The codes of query processing and verification programs are written in Python and Golang based on the B^+-tree structure and the KZG commitments[4].

6.1 Experiment Setting

We describe the detailed experiment configuration. The PCB-tree is built based on the real transaction dataset from Ethereum blockchain. It contains 1,000 blocks with 96,287 transactions, and each transaction is defined as $\langle$*timestamp*, *value*, *from*, *to*$\rangle$, where the *timestamp* is the query period, *value* is the amount of transaction transferred, *from* and *to* are the addresses of sender and receiver respectively.

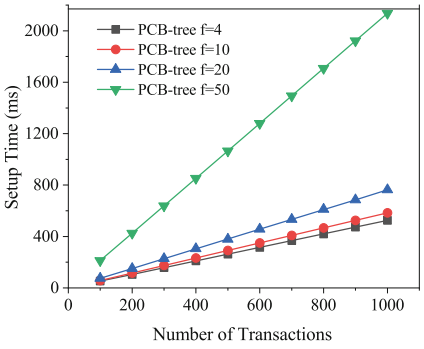


Fig. 4. Setup cost of miner

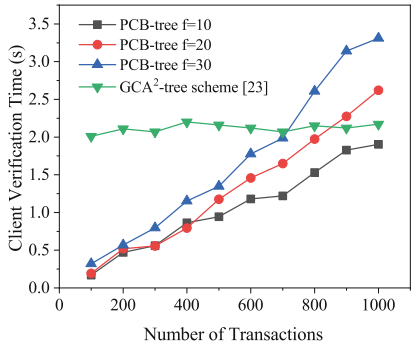


Fig. 5. Verification cost of light client

To evaluate the entire system's performance, we perform four sets of experiments. Firstly, we evaluate the setup cost of ADS construction for the miner. Then, we evaluate the query processing cost of the full node and compare it with the GCA^2-tree [23] which implements a verifiable query scheme using the same accumulator as vChain [19]. Finally, we measure the result verification cost on the light client and the size of the VO.

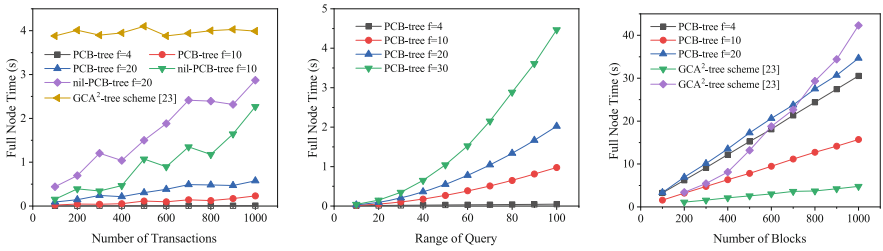
6.2 Experiment Evaluation

Setup Cost. We start with evaluating the construction time of PCB-tree on the miner-side. From Fig. 4, we can learn that the construction time increases

[3] https://infura.io/.
[4] https://github.com/protolambda/go-kzg.

Table 2. Time cost of proof

Fanout	Single process		Batch process (max)	
	Generation	Verification	Generation	Verification
2	0.515 ms	2.451 ms	0.514 ms	2.688 ms
10	0.609 ms	2.728 ms	0.717 ms	5.548 ms
20	1.385 ms	2.813 ms	4.97 ms	13.477 ms
30	2.281 ms	3.027 ms	17.086 ms	28.986 ms
50	7.839 ms	7.109 ms	78.115 ms	96.106 ms



(a) Query Time vs. Different Transactions Number (b) Query Time vs. Range of Query (c) Query Time vs. Different Blocks Number

Fig. 6. Range query performance of full node

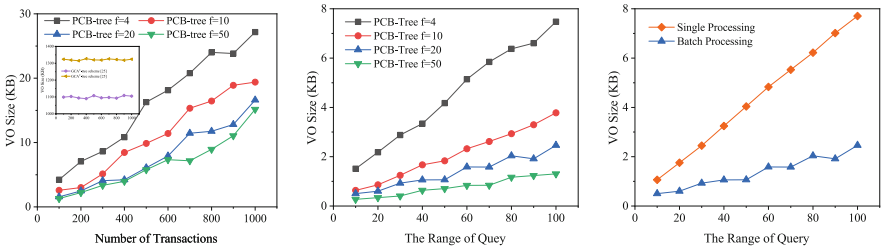
linearly when the number of transactions in a block grows, and as the fanout increases, the construction overhead becomes more expensive, but this ADS construction operation does not affect the performance of blockchain. On the one hand, the current block generation rate of Ethereum is 15s/block, and the average number of transactions per block does not exceed 400. Therefore, the miner can do the construction and mining processes in parallel. On the other hand, the experiment is run a personal computer with a single thread, which is impractical for the miner. Moreover, our PCB-tree ensures the integrity of query results, which is crucial for the verifiable query scheme.

Query Performance. We first test the performance of range queries from different dimensions as shown in Fig. 6 and compare the time cost of queries with the scheme in [23]. Figure 6(a) illustrates the range query performance of full nodes when the number of transactions in a block increases. The full node query performance contains two parties: results query time and proof generation time. It can be seen that the cost of queries increases only linearly with enlarging the transactions number. Meanwhile, its query time increases as the fanout increases. Compared with the single operation, proofs batch aggregation will degrade quickly the query time, where *nil-PCB-tree* represents that batch processing is not used. In theoretical analysis, the complexity of the range query is approximate $\mathcal{O}(log_f n)$. However, it is also necessary to generate proofs for the corresponding results during the query process, which is expensive and is the

Table 3. Comparison of proof size

Scheme	Commitment size	VO format	One VO size	Public parameters	Batch operation
Merkle Tree [9]	32	Hash	$10 \cdot 32 = 320$	–	N
MB-Tree [6]	32	Hash	$3 \cdot 9 \cdot 32 = 864$	–	N
vChain [19]	32	hash,π_v, Attr, Digest	$10 \cdot 288 = 2880$	$64 \cdot 1000 = 64000$	Y
PCB-Tree (this work)	64	π_{pcb}, Attr	$3 \cdot (64 + 64) = 384$	$64 \cdot 10 = 640$	Y

Note: We use a 256-bit group and BN256 elliptical curve for class groups at 128-bit security. We assume the number of transactions is $n = 1000$, the size of attribute messages is 256-bit (An 'Attr' field consists of 2 elements) and 256-bit hashes. We assume the fanout $f = 10$ for PCB-tree and MB-tree and the path height of Merkle tree is 10.



(a) VO size vs. Different Transactions Number (b) VO size vs. Range of Query (c) Batch vs. Single Processing

Fig. 7. Communication cost

primary cost of the full node. Furthermore, with the increase of fanout, the time of proof generation and verification increases which account for the increment of query time. The time cost of proof is shown in Table 2.

We repeat the above experiment while fixing the transactions number to 500 in a block and varying the range of queries. Figure 6(b) shows the changing trend of query time when the range of query increases. As the range of queries, the leaf proof number becomes longer, which accounts for the linear increase in our systems. Figure 6(c) shows the query performance at different block numbers. Each block contains 500 transactions and fixes the range of queries to 15. Since we mainly consider the performance of a single block in this paper, we will execute the single block query algorithm recursively for different blocks.

VO Size. Next, we measure the communication cost between the full node and the light client. Firstly, we theoretically analyze the storage cost of commitments, proof, and public parameters of various schemes, as shown in Table 3. Our public key parameters and VO size are small compared to scheme [19]. In our schemes, the proofs for non-leaf and leaf nodes are $\langle k_i, c_i, \pi_i \rangle$ and $\langle k_i, \pi_i \rangle$ respectively. The batch proofs for non-leaf and leaf nodes are $\langle m(k_i, c_i), r(k), \pi_i \rangle$ and $\langle m(k_i), r(k), \pi_i \rangle$ respectively, where m is the number of elements aggregated, $r(k)$ is the remainder of the polynomial division and the size is 32 bytes. To alleviate communication overhead, we can only send x-axis coordinates of the

elliptic curve points and add additional one-bit messages to record the positive and negative. In this case, the proof size would only be 32 bytes.

Figure 7(a) illustrates the VO size with increasing numbers of transactions. It can be seen that the VO size grows linearly with the number of transactions. However, the order of magnitude of our VO size is *KB*, and the scheme [23] is *MB*. Figure 7(b) and 7(c) shows the VO size with varying query range. The transactions number is fixed to 500 at per block, and the fanout of Fig. 7(c) is 20. We observe that the VO size is small when the tree fanout is small. Meanwhile, the proof aggregation will reduce quickly the communication overhead transferred from the full node to the light client. In contrast, without batch processing, the VO size increases linearly at least 3×. Therefore, based on Fig. 6 and Fig. 7, we conclude that our scheme makes a trade-off between query efficiency and VO size.

Verification Cost. Finally, we evaluate the verification cost at the light client with the number of transactions queried. We mainly discuss the cost of proof verification in this experiment because it is a major overhead for the client. Figure 5 demonstrates that the verification time grows linearly with the transactions number, and the verification time of scheme [23] is a stable horizontal line. However, we discover that the clients generally query transactions they are interested in the recent period, and the experiment shows that when the numbers of transactions that are interested $\leq$800, the client verification efficiency of our scheme is better than scheme [23].

7 Conclusion

In this paper, we study the problem of verifiable query processing and propose an efficient and verifiable Boolean range query scheme for light clients on blockchain databases. Firstly, we developed a novel authenticated data structure, polynomial commitment B^+-tree (PCB-tree). Based on that, we achieve efficient integrity and correctness verification of Boolean and range queries for blockchain light clients. Secondly, our scheme provides a tunable trade-off between query time and communication overhead by autonomously setting the fanout size of the PCB-tree. Thirdly, our scheme can further support batch processing to reduce VO size and verification time. Finally, security analysis and experiment have substantiated that our scheme is secure and efficient.

Acknowledgements. This work is supported by the Key Research and Development Program of Shaanxi (Program No. 2022KWZ-01), the Key Research and Development Program of Shaanxi (No. 2020ZDLGY08-03), and the Fundamental Research Funds for the Central Universities (No. JB211503).

A Pseudo Codes of the PCB-Tree Algorithms

Algorithms 5 and 6 respectively show the query processing of the PCB-tree introduced in Sect. 5.

Algorithm 5: Range Query w.r.t. PCB-tree

```
1  Function RangeQuery(curr, l, u, R, VO):
       Input: Current node curr, Lower bound l, Upper bound u, Results
              set R, Verification object VO
2      if curr is not leaf then
3          left = getPos(curr, l);
4          right = getPos(curr, u);
5          p(x) ← curr.p(x);
6          if left == right then
7              ω_left = CreateWitness(PK, p(x), k_left);
8              add ⟨left, (k_left, v_left), ω_left⟩ to VO;
9              RangeQuery(curr.children_left, l, u, R, VO);
10         else
11             for i=left to right; i += 1 do
12                 ω_i = CreateWitness(PK, p(x), k_i);
13                 add ⟨i, (k_i, v_i), ω_i⟩ to VO;
14             end
15             max = curr_left.max;
16             RangeQuery(curr.children_left, l, max, R, VO);
17             min = curr_right.min;
18             RangeQuery(curr.children_right, min, u, R, VO);
19         end
20     else
21         left = getPos(curr, l);
22         while curr is not None do
23             p(x) ← curr.p(x);
24             for i ← left to curr.size() do
25                 if curr.keys[i] > u then
26                     break;
27                 add ⟨i, curr.keys[i], tx_i⟩ to R;
28                 ω_i = CreateWitness(PK, p(x), k_i);
29                 add ⟨i, (k_i, v_i), ω_i⟩ to VO;
30             end
31             if curr.next is not None then
32                 curr = curr.next;
33                 left = 0;
34             end
35         end
36     end
```

Algorithm 6: Position Search Algorithm

```
1  Function getPos(curr, key):
       Input: The current node curr, The search key key
       Output: The position index z
2      count ← curr.keyNum;
3      z ← -1;
4      if count ≠ 0 then
           // binary search
5          lo ← 0;
6          hi ← entries;
7          while z < 0 do
8              mid ← (lo+hi) // 2;
9              diff ← key-curr.keys[mid];
10             if diff ≤ 0 then
11                 if key-curr.keys[mid-1] > 0 then
12                     z ← mid;
13                 else
14                     z ← -2;
15                 end
16             else
17                 if key-curr.keys[mid+1] ≤ 0 then
18                     z ← mid+1;
19                 else
20                     z ← -3;
21                 end
22             end
23             if z == −2 then
24                 lo ← 0;
25                 hi ← mid-1;
26             else if z == -3 then
27                 lo ← mid+1;
28                 hi ← hi;
29         end
30     else
31         z ← -1;
32     end
33     return z;
```

References

1. Antonopoulos, A.M.: Mastering Bitcoin: Programming the Open Blockchain. O'Reilly Media, Inc. (2017)
2. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive (2020)

3. Cai, C., Xu, L., Zhou, A., Wang, R., Wang, C., Wang, Q.: EncELC: hardening and enriching ethereum light clients with trusted enclaves. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 1887–1896. IEEE (2020)
4. Comer, D.: Ubiquitous b-tree. ACM Comput. Surv. (CSUR) **11**(2), 121–137 (1979)
5. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 177–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
6. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, pp. 121–132 (2006)
7. LongHash: Blockchain big data problem. https://www.longhash.com/en/news/2382/Blockchain's-Big-Data-Problem
8. Matetic, S., Wüst, K., Schneider, M., Kostiainen, K., Karame, G., Capkun, S.: BITE: bitcoin lightweight client privacy using trusted execution. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 783–800 (2019)
9. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_21
10. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2009). https://bitcoin.org/bitcoin.pdf
11. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy, pp. 238–252. IEEE (2013)
12. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_9
13. Qin, K., Hadass, H., Gervais, A., Reardon, J.: Applying private information retrieval to lightweight bitcoin clients. In: 2019 Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 60–72. IEEE (2019)
14. Ramakrishnan, R., Gehrke, J., Gehrke, J.: Database Management Systems, vol. 3. McGraw-Hill, New York (2003)
15. Ruan, P., Chen, G., Dinh, T.T.A., Lin, Q., Ooi, B.C., Zhang, M.: Fine-grained, secure and efficient data provenance on blockchain systems. Proc. VLDB Endow. **12**(9), 975–988 (2019)
16. Shao, Q., Pang, S., Zhang, Z., Jing, C.: Authenticated range query using SGX for blockchain light clients. In: Nah, Y., Cui, B., Lee, S.-W., Yu, J.X., Moon, Y.-S., Whang, S.E. (eds.) DASFAA 2020. LNCS, vol. 12114, pp. 306–321. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59419-0_19
17. Smith, C., Rusnak, A.: Dynamic Merkle B-tree with efficient proofs. arXiv preprint arXiv:2006.01994 (2020)
18. Wiki, E.: Ethereum light client protocol. https://eth.wiki/concepts/light-client-protocol
19. Xu, C., Zhang, C., Xu, J.: vchain: Enabling verifiable Boolean range queries over blockchain databases. In: Proceedings of the 2019 International Conference on Management of Data, pp. 141–158 (2019)
20. Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G.: Authenticated indexing for outsourced spatial databases. VLDB J. **18**(3), 631–648 (2009)
21. Zhang, C., Xu, C., Wang, H., Xu, J., Choi, B.: Authenticated keyword search in scalable hybrid-storage blockchains. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 996–1007. IEEE (2021)

22. Zhang, C., Xu, C., Xu, J., Tang, Y., Choi, B.: Gem^ 2-tree: a gas-efficient structure for authenticated range queries in blockchain. In: 2019 IEEE 35th international conference on data engineering (ICDE), pp. 842–853. IEEE (2019)
23. Zhu, Y., Zhang, Z., Jin, C., Zhou, A.: Enabling generic verifiable aggregate query on blockchain systems. In: 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), pp. 456–465. IEEE (2020)