Anole: A Lightweight and Verifiable Learned-based Index for Time Range Query on Blockchain Systems

Jian Chang, Binhong Li, Jiang Xiao, Licheng Lin, and Hai Jin

National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
`{j_cahng,binhong,jiangxiao,lichenglin,hjin}@hust.edu.cn`

Abstract. Time range query is essential to facilitate a wide range of blockchain applications such as data provenance in the supply chain. Existing blockchain systems adopt the storage-consuming tree-based index structure for better query performance, however, fail to efficiently work for most blockchain nodes with limited resources. In this paper, we propose *Anole*, a lightweight and verifiable time range query mechanism, to present the feasibility of building up a learned-based index to achieve high performance and low storage costs on blockchain systems. The key idea of *Anole* is to exploit the temporal characteristics of blockchain data distribution and design a tailored lightweight index to reduce storage costs. Moreover, it uses a digital signature to guarantee the correctness and completeness of query results by considering the learned index's error bounds, and applies batch verification to further improve verification performance. Experimental results demonstrate that *Anole* improves the query performance by up to 10× and reduces the storage overhead by 99.4% compared with the state-of-the-art vChain+.

Keywords: Blockchain · Time range query · Learned index · Lightweight.

1 Introduction

Blockchain has become a promising distributed ledger technology for multiple parties to engage and share a decentralized tamper-proof database [13]. Blockchain systems record the transactions between these parties in timestamped and chronologically linked data blocks [11]. Time range query is the most fundamental query type on blockchain systems that retrieves data blocks within a given time interval. It has played a pivotal role in supporting trustworthy data provenance and traceability for many crucial applications, such as supply chain [8], smart manufacturing [9], healthcare [12]. For example, blockchain time range query can support efficient and trustworthy contact tracing, vaccine logistics, and donations during the COVID-19 outbreak. Donors hope to query about their donations (Bitcoin or Ethereum, etc.) within a given time interval, so as to know the whereabouts of the donation funds. The client sends Q = ([Addr:2AC03E7F], [2022.06.18, 2022.09.17], [in/out]) to get the transactions for address 2AC03E7F from June 18, 2022, to September 17, 2022.

A lightweight and verifiable time range query mechanism is desirable and important for blockchain systems. While blockchain full nodes (e.g., resource-rich servers) store the ever-growing immutable data blocks, they can retrieve reliable results upon the full history. In practice, however, most users on blockchain are light nodes (e.g., mobile users with constrained resources), who can only rely on full nodes by proceeding with remote queries. Since the full nodes can be malicious in the trustless blockchain, light nodes must further verify the query results to ensure the correctness and completeness. Moreover, the indexing for accelerating query processing will bring in additional and even unacceptable storage overhead [24].

Prior studies [14, 18, 20] have exploited the tree-based index structure for facilitating verifiable blockchain Boolean range query. The authors built a *verification object* (VO) set and reconstructed the Merkle tree root to verify the query results. Since the tree-based indexing query mechanism trades off storing the large-size VO for query performance, it is not affordable for lightweight nodes with limited storage space. Furthermore, it incurs an excessively time-consuming traverse process (i.e., each tree element must be equally traversed), leading to significant performance degradation.

In this paper, we explore an alternative approach towards lightweight and verifiable blockchain time range query. Our key insight is that the data blocks of a blockchain exhibit a temporal distribution by nature, as the time interval between block generation is determined by the underlying consensus protocol. For example, the Bitcoin system generates a block about every ten minutes with the *Proof-of-Work* (PoW) consensus. It allows us to build a regression function between the timestamp (i.e., key) and block height (i.e., the value of the position). Inspired by [19], our design takes the advantage of learned index that supports efficient time range query tailored to the blockchain data.

However, it is challenging to build up an appropriate blockchain learned index that can simultaneously achieve high performance and low storage costs. The first *challenge 1* is how to build a novel index structure based on the temporal distribution of blockchain. The malicious operations in a trustless blockchain system can prohibit the procedure of block generation, such as the selfish mining attacks [2, 10] that delay publication of blocks (i.e., 5% of blocks were generated for more than 30 minutes). The second *challenge 2* is even after we successfully construct a tailored learned index, how to design a lightweight and efficient method to verify the query results? Because of diverse underlying storage models, it is not possible to directly apply the tree-based VO on learned index. The learned index can locate the position of the query data in a list, but it is difficult to map to the position in the *merkle hash tree* (MHT) to get VO. Hence, the conventional MHT-based verification approaches are not suitable for learned index structures.

To address the above challenges, we present *Anole*, a lightweight and verifiable time range query mechanism that explores the use of learned index structures on blockchain systems. It includes three major components: (1) To automatically learn the relationship between the timestamp and block height, *Anole* leverages piecewise linear functions to build a novel layered learned index structure, which can reduce the storage cost by orders of magnitude by storing function argument. (2) *Anole* proposes the aggregate signature that greatly reduces the VO size and alleviates the burden of data transmission in the blockchain network compared to the classical tree-based approaches. (3) We further develop a lightweight batch verification for ensuring the correctness and completeness, while retaining high performance. We have implemented a fully functional, open-sourced query prototype of *Anole*.

In summary, we make the following contributions:

- We propose *Anole*, a novel learned index-based lightweight and efficient mechanism for blockchain time range query. To the best of our knowledge, this is the first layered learned index structure that can automatically bound the query results by temporal distribution patterns. Moreover, it can reduce the storage overhead of building indexes, and improve query performance.
- We also develop two optimizations - aggregation signature and batch verification - which significantly reduce the verification overhead. It not only keeps the VO size lightweight and guarantees the verifiability of query results.
- Experimental results demonstrate that *Anole* significantly outperforms the state-of-the-art vChain+ [18] in terms of storage overhead and query performance.

The rest of this paper is organized as follows. We outline the related work in Section 2, prior to introducing the system overview of Anole in Section 3. Section 4 and Section 5 present the detailed design of layered learned index and lightweight verification process of Anole. Comprehensive evaluation is shown in Section 6. Finally, we conclude this work in Section 7.

2 Related Work

In this section, we review the most relevant range query techniques over traditional database and discuss state-of-the-art blockchain verifiable query mechanisms in Table 1.

Database Range Queries. Considering the impact of data partitioning on large-scale data processing, Yue et al. [22] proposed a time-based partitioning technology to provide range query operations on large-scale trajectory data. To further improve the query efficiency, learned-based techniques for data query are used in database [1,17]. The structure of the FITing-Tree [7] is very similar to the traditional B+ tree, and the difference is that its leaf nodes store the start key and slope of each segment. PGM-index [6] optimized FITing-Tree from data segmentation, insertion, and deletion, which can achieve better query and update time efficiency. ALEX [5] proposed another scheme to support insertion operation in which a newly arrived key keeps the array in order at the predicted position gap. These learned indexes can efficiently support range query operations in traditional databases but fail to offer verifiable queries in a trustless blockchain environment with malicious nodes.

Table 1. The comparison of Anole with existing query mechanisms

Category	Approach	Time	Lightweight		Verification	
		range	Index size	VO size	Correctness	Completeness
Distributed database	FITing-Tree [7]	✓	✓	✗	✓	✗
	PGM-index [6]	✓	✓	✗	✓	✗
	ALEX [5]	✓	✓	✗	✓	✗
Blockchain	LineageChain [15]	⊙	✗	⊙	✓	✓
	GEM^2-Tree [23]	✗	⊙	✗	✓	✓
	P^{2}B-Trace [14]	⊙	✗	⊙	✓	⊙
	LVQ [4]	✗	⊙	⊙	✓	✓
	vChain+ [18]	✓	⊙	⊙	✓	✓
	Anole	✓	✓	✓	✓	✓

NOTE: ✓ : support ⊙: poor support ✗ : not support

Blockchain Verifiable Queries. Efficient query and processing of a large number of time series data have attracted unprecedented attention from both industry and academia [21]. Shao et al. [16] presented an authentication range query scheme based on *Trusted Execution Environment* (TEE). However, due to the limited secure memory space, existing TEEs cannot easily handle large-scale applications. Besides, GEM^2-tree [23] designed a two-level index structure, which is gas-efficient and effective in supporting authenticated queries. LVQ [4] presented a new Bloom filter based on sorted Merkle Tree to achieve lightweight verifiable but not support time range query. Unfortunately, the maintenance cost of the tree-based *authenticated data structure* (ADS) is relatively heavy for light nodes to support the verification procedure. Moreover, LineageChain [15] provided a new skip list index to support efficient provenance. To overcome the practical problem of public key management in vChain, vChain+ [18] proposed a new *sliding window accumulator* (SWA) to reduce the public key storage overhead of accumulators. However, their design takes up a large amount of storage space for VO and computing overhead, which requires high node configuration.

3 System Overview

Fig. 1 shows the overview of *Anole* system consisting of three actors: miners, full nodes, and clients. The miners, responsible for packaging data into blocks and appending new blocks to the blockchain, are considered trusted third parties because of the rewarding scheme of the blockchain system. The full nodes store both block headers and data, responsible for processing the client's query requests. When the clients send a query request, the full nodes take advantage of the learned index to find the location of request data, and then return the query result and corresponding digital signature as VO for the clients to verify. The clients, who store only block headers, use the public key and VO to check the completeness and correctness of the query result.

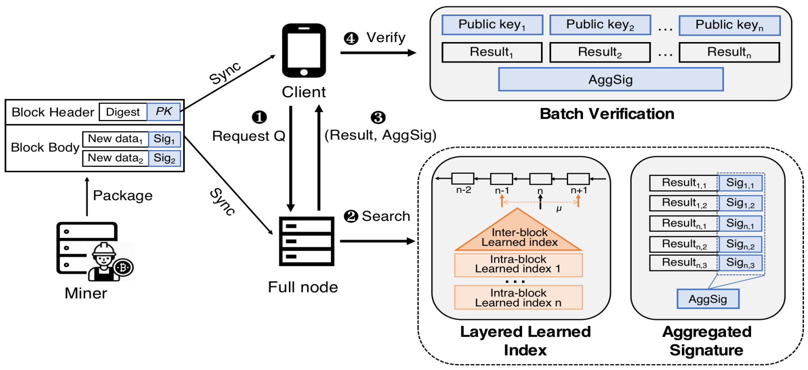


Fig. 1. The system overview of *Anole*

Suppose a client C submits a request Q to a full node for retrieving the transactions during last two weeks on blockchain (Step ❶ in Fig. 1). To ensure query efficiency, the full node utilizes learned index for retrieval, that is, the block height range of the element is quickly located through the inter-block learned index, and the query results that meet the conditions are searched through the intra-block learned index (step ❷). Multiple digital signatures are combined into one aggregated signature to reduce the overall VO size. After query execution, the full node assembles a tuple, including the result and the *aggregated signature* (AggSig), and sends it to the client (Step ❸). Upon receiving all the results and aggregated signature from the full node, the client obtains the corresponding public keys by synchronizing the block header to verify the returned results and VO in batch (Step ❹).

4 Learned Index-based Time Range Queries

In this section, we propose a layered learned index to meet *challenge 1*, which captures mapping relationships between timestamps and block height. Furthermore, we discuss efficient query execution and theoretical analysis of error bound.

4.1 Layered Learned Index

Layer 1: Inter-block Learned Index. The main idea of our method is to extract the mapping relationship between the data through a piecewise linear function. We propose a dynamic piecewise linear regression algorithm based on the distribution of timestamp and block height, which is linear in run time. *Anole* introduces error bound to ensure that all the data of the learned index can be retrieved, including the occurrence of outliers that deviate from the normal distribution. More specifically, we define the error bounds for dynamic piecewise linear regression by three parallel lines: the start function that a line formed by two starting points, the upper boundary function that the regression function plus the error bound, and the lower boundary function that the regression function minus the error bound as shown in Fig. 2. Points 1 and 2 are the two basic points that form the starting function. Point 3 is inside the two boundaries, and the current regression does not need to be updated. As point 4 is outside the updated regression boundary (i.e., the actual value of point 4 is under the lower boundary based on the error bound μ), a new piecewise regression function will be generated. The combination of these two boundary functions gives the edges of the regression function. Intuitively, the two boundary functions represent a sequence of feasible linear regressors around the beginning of the regression function.

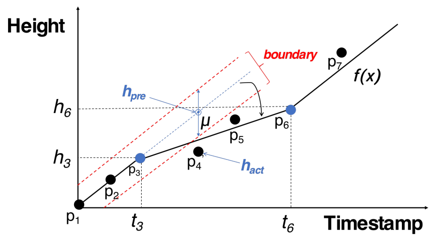


Fig. 2. The dynamic regression of inter-block learned index

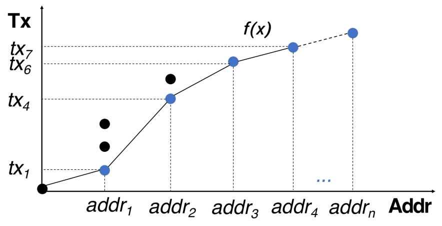


Fig. 3. The mapping of intra-block learned index

Layer 2: Intra-block Learned Index. One block usually contains numerous transactions (e.g., bitcoin averages over 1,000 transactions per block). For an active address, in addition to having multiple transactions between different blocks, there may also be multiple transactions within a block. Based on this observation, we construct an intra-block learned index to optimize the query time in block. The intra-block index is constructed by blockchain miners in an aggregated manner based on identical address transactions. Fig. 3 shows the block with an intra-block index. It sorts and aggregates the transactions corresponding to each address, and the first transaction of the identical address is used as the aggregation point (e.g., tx_1 for $addr_1$, and tx_4 for $addr_2$). Then, the intra-block learned index is constructed based on the sorted transactions and different aggregation points. In particular, since the amount of data in a single block is small and does not need to be updated, the error bound of intra-block learned index can be set to 0 to achieve precise positioning.

4.2 Efficient Query Execution

In this section, we start by considering a particular timestamp and focusing on the point query with learned index for ease of illustration. Then we extend it to the time range query condition to show how to process time range query requests efficiently. Anole maintains the parameters of each segment, including the starting point, the slope, and the intercept. The timestamp value is an increment property, hence we use a variant of the binary search algorithm to quickly search the segment where the key value is located. The time complexity is $O(log_2(n))$ of searching for the specific segment, where n is the length of segment list. Once the segment is found with the corresponding timestamp, we can locate the block height of the given timestamp by calculating the function in this segment. Recall that when creating a piecewise linear function, the actual position of the key is kept in a range (i.e., error bound μ) from the position calculated by the regression function. According to the slope and intercept parameter of segment s, we can obtain the predicted position of the given timestamp t by calculating the following equation:

$$pred_h(t) = t * s.slope + s.intercept \tag{1}$$

The true location of the elements can be restricted to the error bound once the learned index is constructed by dynamic piecewise linear function. Consequently, after the predicted position is obtained by calculation, sequential traversal is used to perform a local retrieval for the blocks within the error bound. The true position of the given timestamp t and error bound μ is calculated by the following equation:

$$true_h(t) \in [pred_h(t) - \mu, pred_h(t) + \mu] \tag{2}$$

Time range query is a special case of range query that has the subsidiary conditions. It requires to check whether each item is within a particular given time range as shown in Algorithm 1. Hence, unlike point query, for a time range query, the condition has a great impact on the total running time and the result size. The main difference between the two query types is that the time range query requires finding two endpoints of the given time range. Since the segment satisfies the given error bound, the overhead of finding the key within the segment can be restricted. To be more precise, the time complexity of searching keys within the bound is $O(1 + 2 * \mu)$. The linear function either points to the store key consecutively in the same segment or exists in adjacent segments, where the segmented points are sorted by value. Therefore, Anole can first simply start the scan at the start point position of the time range within the error bound, and then traverse the adjacent segments to the end point of the range.

Algorithm 1: Time Range Query

```
input  : $Q = (addr, < t_1, t_2 >)$
output: $R = (TxSet, VO)$
1  $(seg_1, seg_2) \longleftarrow VBS(t_1, t_2)$;
2  $pre\_h_i \longleftarrow seg_i.slope * t_i + seg_i.intercept \quad (i = 1, 2)$;
3  for $h$ in $(pred\_h_1 - error)..(pred\_h_2 + error)$ do
4      $Tx.id \longleftarrow intra\_fn(addr)$;
5      if $Tx$ exist then
6          $TxSet = TxSet.append(Tx)$;
7          $VO = aggr(Tx.sign)$;
8      else
9          $TxSet = prior(Tx.id) + next(Tx.id)$;
10         $VO = prior(Tx.id).sign + next(Tx.id).sign$
11     end
12 end
13 return $R = (TxSet, VO)$
```

4.3 Theoretical Analysis

The error bound has an impact on the query efficiency and the size of the index. Inevitably, the following question naturally arises: how to choose the error bound? To deal with this trade-off, we define an assessment model to choose a "suitable" error bound during the learned index construction. There are two essential indicators that can be optimized regarding the error bound: the query performance impact on the system (i.e., query latency) and the storage overhead of the system (i.e., index size).

The value of error bound affects the segment's generation and the searching precision (i.e., the large error bound has fewer segments produced and lower precision). We let N_μ denote the segment's amount. The query latency for computation estimated by the error bound μ can be calculated by the following equation, where α is the number of function parameters, β is the intra-block transaction data, and c is the system delay of the instruction on given hardware (e.g., 10ns).

$$Latency(\mu) = c * (log_2(N_\mu) + \alpha + 2\mu * \beta) \tag{3}$$

For a given error bound μ, we can evaluate the storage cost of the learned index (in Byte) using the following equation. The first part of the formula is the size of the index parameter (i.e., the slope and intercept parameter, each 2 bytes), and the second part is the space occupied by the digital signature within the error bound (64 bytes/block), which is discussed in the verification section.

$$Storage(\mu) = N_\mu * \alpha * 2B + 2\mu * 64B \tag{4}$$

Based on these two cost estimation formulas, it can be obtained that the minimum storage for the learned index that meets a specific latency demand $L(ms)$ or the minimum error bound for the learned index that satisfies a given storage budget $S(bytes)$. Therefore, the most suitable μ is given by the following expression, where E denotes a set of possible error bounds (e.g., $E = \{1, 2, 5, 10\}$).

$$\mu = argmin \begin{cases} Storage(\mu) & | \quad Latency(\mu) \leq L \\ Latency(\mu) & | \quad Storage(\mu) \leq S \end{cases}, \quad \mu \in E \tag{5}$$

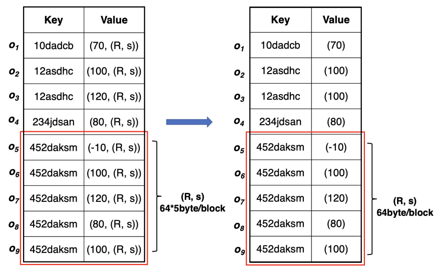
5 Lightweight Verification

In this section, we propose a lightweight verification scheme based on the learned index and digital signatures to meet *challenge 2*, which omits the step of re-traversing in MHT and enables dynamic half-aggregation to reduce the VO size. Furthermore, we optimize the digital signatures to achieve efficient batch verification on the basis of ensuring security to remedy the defect of long verification time.

5.1 Lightweight VO Design

Aggregated Signature. We divide the aggregated signatures into two parts: inter-block signature aggregation and intra-block signature aggregation. Intra-block signature aggregation is done by miners when packing blocks. As in the example shown in Fig. 4, miners bundle transactions with the same address, e.g., join all transactions with an address of $452daksm$, $O = o_5||o_6||o_7||o_8||o_9$. Then miners sign the object O to obtain the signature (R, s). In this way, it greatly reduces the size of the verification object in a block, as each block returns only one signature (R, s) for a certain address. To further reduce the size of VO, Anole proposes a dynamic inter-block half-aggregation technique. As the block height will change with the time range, Anole assigns the inter-block signature aggregation to the full node. With this method, the full node can dynamically aggregate signatures according to the query request and not return redundant data for verification. The implementation is shown in Fig. 5, where λ is the security parameter, g is the generator of a cyclic group G, (Pk, sk) is the keypair where Pk is the public key and sk is the private key, σ is the signature which can be decomposed into (R, s), and m is the message. It is worth noting that the coefficient L can effectively prevent a malicious full node from tampering with query data or signatures, as it commits to each signature, message, and public key.

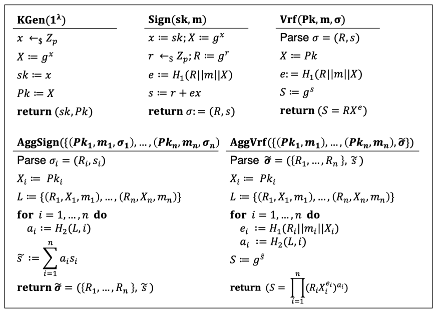
Batch Verification. Compared with the MHT-based authentication method, the digital signature eliminates unnecessary data and greatly reduces the size of the verification object, but it pays a price in terms of verification time. To solve this problem, we design a batch verification method based on the Straus algorithm [3] to speed up the verification. The basic authentication method is to verify the signature of query results according to the aggregated signature by



	Key	Value
o_1	10dadcb	(70, (R, s))
o_2	12asdhc	(100, (R, s))
o_3	12asdhc	(120, (R, s))
o_4	234jdsan	(80, (R, s))
o_5	452daksm	(-10, (R, s))
o_6	452daksm	(100, (R, s))
o_7	452daksm	(120, (R, s))
o_8	452daksm	(80, (R, s))
o_9	452daksm	(100, (R, s))

	Key	Value
o_1	10dadcb	(70)
o_2	12asdhc	(100)
o_3	12asdhc	(100)
o_4	234jdsan	(80)
o_5	452daksm	(-10)
o_6	452daksm	(100)
o_7	452daksm	(120)
o_8	452daksm	(80)
o_9	452daksm	(100)

Fig. 4. Example of intra-block aggregated signatures



```
KGen(1^λ)
    $x \leftarrow_\$ Z_p$
    $X := g^x$
    $sk := x$
    $Pk := X$
    return $(sk, Pk)$

Sign(sk, m)
    $x := sk; X := g^x$
    $r \leftarrow_\$ Z_p; R := g^r$
    $e := H_1(R||m||X)$
    $s := r + ex$
    return $\sigma := (R, s)$

Vrf(Pk, m, σ)
    Parse $\sigma = (R, s)$
    $X := Pk$
    $e := H_1(R||m||X)$
    $S := g^s$
    return $(S = RX^e)$

AggSign({(Pk_1, m_1, σ_1), ..., (Pk_n, m_n, σ_n)
    Parse $\sigma_i = (R_i, s_i)$
    $X_i := Pk_i$
    $L := \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$
    for $i = 1, \ldots, n$ do
        $a_i := H_2(L, i)$
    $\widetilde{s} := \sum_{i=1}^{n} a_i s_i$
    return $\widetilde{\sigma} = (\{R_1, \ldots, R_n\}, \widetilde{s})$

AggVrf({(Pk_1, m_1), ..., (Pk_n, m_n), σ̃})
    Parse $\widetilde{\sigma} = (\{R_1, \ldots, R_n\}, \widetilde{s})$
    $X_i := Pk_i$
    $L := \{(R_1, X_1, m_1), \ldots, (R_n, X_n, m_n)\}$
    for $i = 1, \ldots, n$ do
        $e_i := H_1(R_i||m_i||X_i)$
        $a_i := H_2(L, i)$
    $S := g^{\widetilde{s}}$
    return $(S = \prod_{i=1}^{n} (R_i X_i^{e_i})^{a_i})$
```

Fig. 5. The scheme of inter-block aggregated schnorr signature

the following equation:

$$g^{\widetilde{s}} = \prod_{i=0}^{n} (R_i X_i{}^{e_i})^{a_i} \tag{6}$$

We analyze the computation overhead of each step in verification and observe that exponentiation in a cyclic group is time-consuming. To deal with this problem, we combine verification with the Straus algorithm to reduce the time overhead of exponentiation. To describe more formally, we transform the right side of the verification equation into the expression:

$$\prod_{i=0}^{n} (R_i X_i{}^{e_i})^{a_i} = \prod_{i=0}^{2n} M_i^{t_i} \tag{7}$$

The verification algorithm performs the following. First, we choose a radix 2^c for the algorithm to compute. Generally, $c = 5$ is appropriate for 256-bit scalars. Second, we precompute the quantity $M_i, 2M_i, ..., (2^c - 1)M_i$ to reduce the cost of subsequent calculations. Third, we recursively compute $\lfloor t_i/2^c \rfloor$ until $t_i = 0$ and record the remainder $N_j = \prod_{i=0}^{n} M_i^{t_i \ mod \ 2^c}$, where j is the number of divisions performed. Finally, we can recursively compute the result R by $R_{j-1} = R_j{}^{2^c} N_j$ until $j = 1$. In this way, the total cost of identifying the forgeries among n signatures at a 2^b security level is compressed from roughly $2^b n$ multiplications to roughly $(2 + 8/lgb)nb$ multiplications.

5.2 Verifiable Query Processing

In this section, we focus on how to generate the verification object for clients to verify the completeness and correctness of query results. We use concrete examples to illustrate the security of the validation process. The query type of Anole is the time range query for a specific address, which is in the form of $Q =< [t_1, t_2], key >$.

Completeness. The completeness proof generated by full nodes consists of two aspects: timestamp and the key of data. For timestamp, the full node returns the left and right boundaries to prove that the query results contain all blocks satisfied the requirement for timestamp. To process the query request, the full node uses the inter-block learned index to locate the block heights (h_1, h_2) corresponding to t_1 and t_2 respectively. Then, check the timestamp of the block height in the interval $[h_1 - \mu - 1, h_1 + \mu + 1]$ and $[h_2 - \mu - 1, h_2 + \mu + 1]$, where μ is the error bound of the intra-learned index. In this way, the full node can easily find the left and right boundaries (h_l, h_r) and add them to VO. Note that h_l and h_r are the maximum block height with the timestamp smaller than t_1 and the minimum block height with the timestamp larger than t_2. Fig. 6 depicts an example of completeness proof for timestamp when the time range query $Q =< [20210513, 20210514], addr >$. According to the calculation results in the interval $2([h_1 - 2, h_1 + 2], \mu = 1)$, we find the maximum block height 114503 as left boundary h_l, of which the timestamp is smaller than 20210513. We can find h_r in the same way and the completeness proof for the timestamp is $[h_l = 114503, h_r = 114508]$.

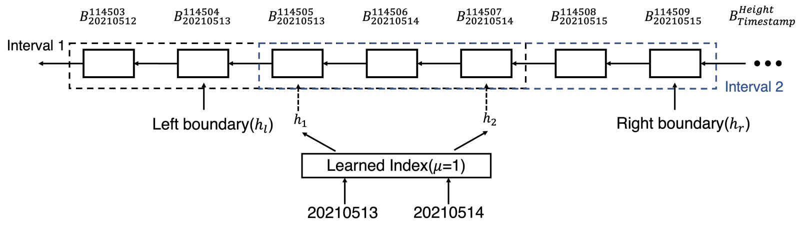


Fig. 6. The completeness proof of timestamp

For the part of key (i.e., address), we divide the completeness proof into two aspects: existence proof and inexistence proof. Existence proof should be able to prove that all the query results are exactly existed inside the block and no key-related transactions are omitted. Since miners are considered as trusted third parties, the aggregated signature (R_i, s_i) that miners generate can be used as the existence proof for the transactions in block i.

For the inexistence proof, we also use the boundary determination principle to prove the key is inexistent in a block. To generate the inexistence proof, we require miners to add an additional attribute p to each transaction when they sort the block data and package the block. The attribute p represents the position of the transaction in the block. When the client queries for a particular transaction, the full node uses the learned index to find the position p_1, where it would be if it exists. As the transactions are sorted lexicographically during block packaging, the full node only returns the corresponding signatures of transactions with positions $p_1 - 1$ and p_1 as boundaries (i.e., the value of key_{p_1} is greater than the value of key_{query}). To keep the VO lightweight, we only return a single signature and transaction for each boundary as inexistence proof.

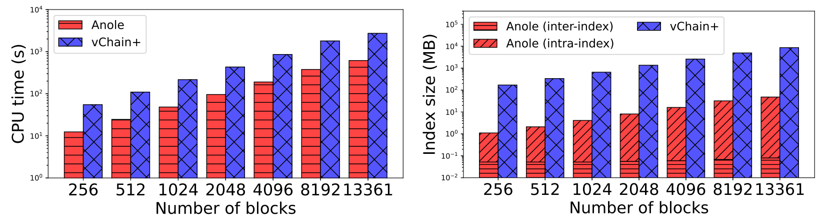
Correctness. Correctness requires that all the query results are satisfied and correct. Consider the VO is $< h_1, h_2, s_{sum}, (R_{addr})_{height} * n) >$. To verify the correctness of query results, the client checks the timestamps of the block in $[h_1, h_2]$ and the key of query results to ensure query results are satisfied. Then, the client can achieve batch verification by calculating the equation 6. Due to the uncontrollable discrete logarithm problem and collision resistance of hash functions, it is hard for full nodes to forge signatures, especially when the private key of signature is not known. It means the query results are correct if the verification is passed.

6 Evaluation

The public Bitcoin data set is used in the experiments, which is extracted from the Bitcoin system from June 18, 2022, to September 17, 2022. It contains 13361 blocks, and the transaction is reorganized as <block height, address, in/out, amount, timestamp>. The miners sign the data and store the public key in the block header (32 bytes). We perform experiments on a server Intel Xeon (Ice Lake) Platinum 8369B with 3.5GHz CPUs, running CentOS 7.6 with 16 cores and 32 GB memory, and the Anole system is programmed in Rust language.

6.1 Index Construction Cost

Fig.7 reveals the index construction overhead and block generation for the miner with the different number of blocks, involving the CPU running time and the storage cost of index. In Anole, the number of blocks is set from 256 to 13361 with the error bound as 2. For vChain+, we set the same block size, and the parameters of fanout and time window for SWA-B+-tree are set to 4 and 2 respectively. The error bound of learned index is set to 2 according to the impact analysis in the following section. From Fig.7(a), we can see that the running



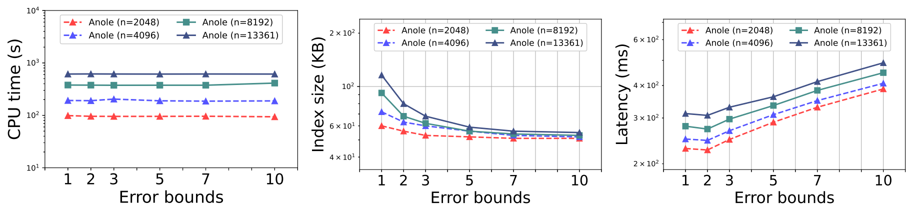
(a) CPU time (b) Index size

Fig. 7. The comparison of index construction cost

time of index construction in Anole is shorter than vChain+ and less than 4 times during the number of blocks is under 13361. Beyond that, Anole yields a smaller index structure compared with vChain+, as shown in Fig.7(b). This is not unexpected because the accumulator employed in vChain+ takes a lot of storage space compared with the function parameters used in Anole to realize data location.

6.2 Impact of Error Bounds

In the following, we assess the effects of various error bounds on the CPU processing time, index size, and latency performance of point queries. We conduct the scalability experiments and utilize the data set from 2048 to 13361 blocks. Fig.8(a) and Fig.8(b) show the performance of CPU time and index size with error bounds varied from 1 to 10. It shows that the CPU time remains almost constant as the error bound increases, and the index size degrades from fast to slow with the increase of error bound. Next, we evaluate the impact of error bounds on latency. Fig.8(c) reveals the query latency of different block heights and the error bound varies from 1 to 10 with the maximum of blocks at 13361. We can observe that the query latency oscillates slightly when the error bound is less than 3, and then increases with the error bound because more block traversals are needed within a large error bound.

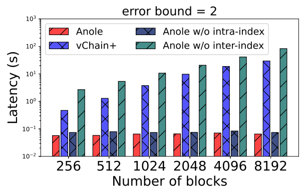


(a) CPU time (b) Index size (c) Latency

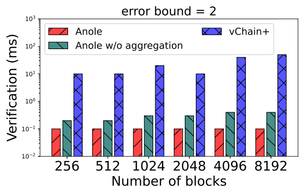
Fig. 8. The impact of error bounds (with the different number of blocks)

6.3 Query Performance

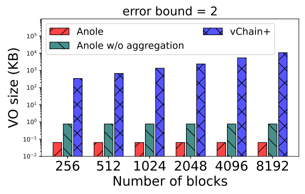
Fig.9 and Fig.10 show the point and time range query performance of Anole during the block height from 1024 to 13361 and the time range from 4 to 24 hours. Three types of query metrics, including query latency, verification time, and VO size are compared. Anole keeps the point query latency within 0.1 seconds as the block height increases. It has 1/100 latency compared with Anole w/o inter-block learned index and 1/10 lower than vChain+ in Fig. 9(a), respectively. When processing time range queries, we observe that Anole and Anole with inter-block index greatly exceed vChain+ when the time range is under 8 hours in Fig. 10(a). The reason is that the inter-block learned index can quickly locate the time range, and combine the intra-block index to find elements. As the verification process for Anole in Fig.9(b) and Fig.9(c), the aggregated signature is returned when the query result exists, which reduces the verification time and VO size. In vChain+, the verification object is generated by ACC.Prove, which includes lots of set operations. The reason we can summarize is that the size of digital signature in Anole is smaller than the tree-based ADS generated by vChain+. As shown in Fig.10(b) and Fig.10(c), when the time range is within 24, the verification time and VO size of Anole are under 0.05s and 5KB, but these performance indicators of vChain+ are 0.6s and 17MB, respectively. We also experiment the throughput of Anole w/o aggregation, which is 70% of Anole as the large VO size with single signature leads to heavy network overhead.



(a) Latency

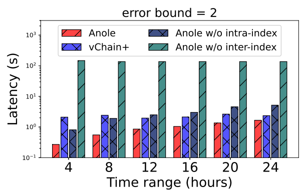


(b) Verification time

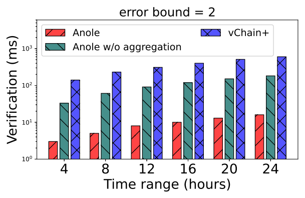


(c) VO size

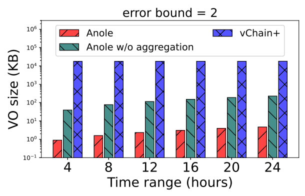
Fig. 9. The comparison of point query performance



(a) Latency



(b) Verification time



(c) VO size

Fig. 10. The comparison of range query performance

7 Conclusion

We present *Anole*, the first lightweight and verifiable learned index-based blockchain time range query mechanism. *Anole* incorporates three key designs: (1) the layered learned index that captures the dynamic temporal distribution of data blocks in the untrusted blockchain environment; (2) the aggregated digital signature technique to reduce the size of the returned verifiable objects; and (3) the design of batch verification to speed up verification while guaranteeing the integrity and correctness of query results. Evaluation of our *Anole* prototype demonstrates that it achieves 10× average speedup and significantly reduces the storage overhead by 99.4%, in comparison with the state-of-the-art vChain+. We hope that our first step in introducing the learned index structures in blockchain query will seed the ground for further exploration on this topic.

Acknowledgement. This work was supported by National Key Research and Development Program of China under Grant No. 2021YFB2700700, Key Research and Development Program of Hubei Province No. 2021BEA164, National Natural Science Foundation of China (Grant No. 62072197), Key-Area Research and Development Program of Guangdong Province No.2020B0101090005, Knowledge Innovation Program of Wuhan-Shuguang. Jiang Xiao is the corresponding author.

References

1. Bi, W., Zhang, H., Jing, Y., He, Z., Zhang, K., Wang, X.: Learning-based optimization for online approximate query processing. In: Proceedings of the 2022 International Conference on Database Systems for Advanced Applications (DASFAA). pp. 96–103 (2022)
2. Bissias, G., Levine, B.: Bobtail: Improved blockchain security with low-variance mining. In: Proceedings of the 2020 Network and Distributed System Security (NDSS) Symposium. pp. 1–16 (2020)
3. Chen, C., Chen, X., Fang, Z.: Addition chains of vectors (problem 5125). American Mathematical Monthly **70**(1), 806–808 (1964)
4. Dai, X., Xiao, J., Yang, W., Wang, C., Chang, J., Han, R., Jin, H.: Lvq: A lightweight verifiable query approach for transaction history in bitcoin. In: Proceedings of the 40th International Conference on Distributed Computing Systems (ICDCS). pp. 1020–1030 (2020)
5. Ding, J., Minhas, U., Yu, J., Wang, C., Do, J., Li, Y., Zhang, H., Chandramouli, B., Gehrke, J., Kossmann, D., Lomet, D.: Alex: an updatable adaptive learned index. In: Proceedings of the 2020 International Conference on Management of Data (SIGMOD). pp. 969–984 (2020)
6. Ferragina, P., Vinciguerra, G.: The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. In: Proceedings of the 2020 International Conference on Very Large Data Bases (VLDB). pp. 1162–1175 (2020)
7. Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R., Kraska, T.: Fiting-tree: A data-aware index structure. In: Proceedings of the 2019 International Conference on Management of Data (SIGMOD). pp. 1189–1206 (2019)
8. Han, R., Xiao, J., Dai, X., Zhang, S., Sun, Y., Li, B., Jin, H.: Vassago: Efficient and authenticated provenance query on multiple blockchains. In: Proceedings of the 40th International Symposium on Reliable Distributed Systems (SRDS). pp. 132–142 (2021)
9. Hewa, T., Braeken, A., Liyanage, M., , Ylianttila, M.: Fog computing and blockchain-based security service architecture for 5g industrial iot-enabled cloud manufacturing. IEEE Transactions on Industrial Informatics (TII) **18**(10), 7174–7185 (2022)

10. Hou, C., Zhou, M., Ji, Y., Daian, P., Tramer, F., Fanti, G., Juels, A.: Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. In: Proceedings of the 2021 Network and Distributed System Security (NDSS) Symposium. pp. 1–18 (2021)
11. Jin, H., Xiao, J.: Towards trustworthy blockchain systems in the era of 'internet of value': Development, challenges, and future trends. Science China Information Sciences (SCIS) **65**(153101), 1–11 (2022)
12. Liu, L., Li, X., Au, M., Fan, Z., Meng, X.: Metadata privacy preservation for blockchain-based healthcare systems. In: Proceedings of the 2022 International Conference on Database Systems for Advanced Applications (DASFAA). pp. 404–412 (2022)
13. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), https://bitcoin.org/bitcoin.pdf
14. Peng, Z., C. Xu, H.W., Huang, J., Xu, J., Chu, X.: P^{2}b-trace: Privacy-preserving blockchain-based contact tracing to combat pandemics. In: Proceedings of the 2021 International Conference on Management of Data (SIGMOD). pp. 2389–2391 (2021)
15. Ruan, P.C., Chen, G., Dinh, T.T.A., Lin, Q., Ooi, B.C., Zhang, M.H.: Fine-grained, secure and efficient data provenance on blockchain systems. In: Proceedings of the 2019 International Conference on Very Large Data Bases (VLDB). pp. 975–988 (2019)
16. Shao, Q., Pang, S., Zhang, Z., Jing, C.: Authenticated range query using sgx for blockchain light clients. In: Proceedings of the 2020 International Conference on Database Systems for Advanced Applications (DASFAA). pp. 306–321 (2020)
17. Vaidya, K., Chatterjee, S., Knorr, E., Mitzenmacher, M., Idreos, S., Kraska, T.: Snarf: a learning-enhanced range filter. In: Proceedings of the 2022 International Conference on Very Large Data Bases (VLDB). pp. 1632–1644 (2022)
18. Wang, H., Xu, C., Zhang, C., Xu, J.L., Peng, Z., Pei, J.: vchain+: Optimizing verifiable blockchain boolean range queries (technical report). In: Proceedings of the 2021 International Conference on Management of Data (SIGMOD). pp. 1–14 (2021)
19. Wu, N., Xie, Y.: A survey of machine learning for computer architecture and systems. ACM Computing Surveys (CSUR) **55**(3), 1–39 (2022)
20. Xu, C., Zhang, C., Xu, J.L.: vchain: Enabling verifiable boolean range queries over blockchain databases. In: Proceedings of the 2019 International Conference on Management of Data (SIGMOD). pp. 141–158 (2019)
21. Yagoubi, D., Akbarinia, R., Masseglia, F., Palpanas, T.: Massively distributed time series indexing and querying. IEEE Transactions on Knowledge and Data Engineering (TKDE) **32**(1), 108–120 (2018)
22. Yue, Z., Zhang, J., Zhang, H., Yang, Q.: Time-based trajectory data partitioning for efficient range query. In: Proceedings of the 2018 International Conference on Database Systems for Advanced Applications (DASFAA). pp. 24–35 (2018)
23. Zhang, C., Xu, C., Xu, J., Tang, Y., Choi, B.: Gem^2-tree: A gas-efficient structure for authenticated range queries in blockchain. In: Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE). pp. 842–853 (2019)
24. Zhang, H., Andersen, D., Pavlo, A., Kaminsky, M., Ma, L., Shen, R.: Reducing the storage overhead of main-memory oltp databases with hybrid indexes. In: Proceedings of the 2016 International Conference on Management of Data (SIGMOD). pp. 1567–1581 (2016)