



# vChain: A Blockchain System Ensuring Query Integrity

Haixin Wang, Cheng Xu, Ce Zhang, Jianliang Xu  
Hong Kong Baptist University  
{hxxwang, chengxu, cezhang, xujl}@comp.hkbu.edu.hk

## ABSTRACT

This demonstration presents vChain, a blockchain system that ensures query integrity. With the proliferation of blockchain applications and services, there has been an increasing demand for querying the data stored in a blockchain database. However, existing solutions either are at the risk of losing query integrity, or require users to maintain a full copy of the blockchain database. In comparison, by employing a novel verifiable query processing framework, vChain enables a lightweight user to authenticate the query results returned from a potentially untrusted service provider. We demonstrate its verifiable query operations, usability, and performance with visualization for better insights. We also showcase how users can detect falsified results in the case that the service provider is compromised.

## ACM Reference Format:

Haixin Wang, Cheng Xu, Ce Zhang, Jianliang Xu. 2020. vChain: A Blockchain System Ensuring Query Integrity. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20)*, June 14–19, 2020, Portland, OR, USA. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3318464.3384682>

## 1 INTRODUCTION

Blockchains enable mutually distrusting parties to maintain a common transaction ledger without a central authority [1, 2, 3]. From the database perspective, a blockchain can be seen as a database storing a large collection of timestamped data records. With the wider adoption of blockchains for data-intensive applications such as finance, healthcare, and supply chains, there has been an increasing demand from users to query the data stored in a blockchain database [4, 5]. For example, in the Bitcoin network, users may want to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

© 2020 Association for Computing Machinery.

ACM ISBN 978-1-4503-6735-6/20/06...\$15.00

<https://doi.org/10.1145/3318464.3384682>

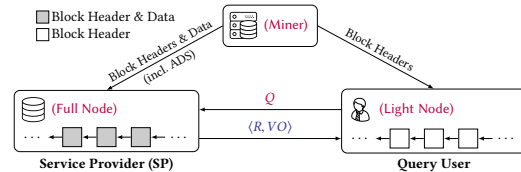


Figure 1: System Model of vChain

find the transactions that satisfy a set of query predicates, such as “*Transaction Fee*  $\geq$  \$50” and “ $\$0.99M \leq$  *Total Output*  $\leq$  \$1.01M”.

While SQL-like search engines have been developed for blockchain databases, the existing solutions rely on a central party who can faithfully execute user queries based on a materialized view of the blockchain database [6, 7]. Unfortunately, such solutions are at the risk of losing query integrity in a distrustful environment where the central party can be malicious or compromised. Alternatively, users can maintain a full copy of the entire blockchain database and query the data locally. However, that is impractical to ordinary users as it requires considerable storage, computing, and bandwidth resources.

To tackle the shortcomings of the existing solutions, we develop vChain, a blockchain system that ensures query integrity by employing a novel *verifiable* query framework proposed in our prior work [1]. As shown in Fig. 1, in vChain, a query user is only required to keep track of the block headers as a light node; the queries are instead outsourced to a full node in the blockchain network, which serves as a *service provider* (SP). Although the SP might be untrusted, the query user is able to authenticate the query results by checking an additional *verification object* (VO). The VO is computed by the SP with the help of a carefully designed *authenticated data structure* (ADS) embedded in the block headers. To further improve the performance, several indexed batch verification techniques are also developed. To the best of our knowledge, vChain is the first blockchain system that supports query processing with integrity assurance. It is our purpose in this demonstration to show the usability and performance of the vChain system. In particular, we build a visualization module for attendees to interact with vChain and gain first-hand insights of integrity-assured query processing.

The rest of the demonstration proposal is organized as

follows. Section 2 elaborates the techniques that enable verifiable queries over blockchain databases. Section 3 overviews the vChain prototype system. The interface of vChain and the demonstration details are presented in Section 4.

## 2 TECHNICAL BACKGROUND

We give a brief introduction to the building blocks of vChain. A tuple  $\langle t_i, V_i, W_i \rangle$  denotes an object stored in the blockchain database, where  $t_i$  is the timestamp of the object,  $V_i$  is a multi-dimensional vector that represents one or more numerical attributes, and  $W_i$  is a set-valued attribute. The Boolean range query we consider is in the form of  $q = \langle [t_s, t_e], [\alpha, \beta], \Upsilon \rangle$ , where  $[t_s, t_e]$  is a temporal range selection predicate for the time period,  $[\alpha, \beta]$  is a multi-dimensional range selection predicate for the numerical attributes, and  $\Upsilon$  is a monotone Boolean function on the set-valued attribute. To answer  $q$ , the SP returns all objects such that  $\{o_i = \langle t_i, V_i, W_i \rangle \mid t_i \in [t_s, t_e] \wedge V_i \in [\alpha, \beta] \wedge \Upsilon(W_i) = 1\}$ .

### 2.1 ADS Generation and Query Processing

As mentioned earlier, vChain augments each block with an additional ADS. The design of the ADS is a key issue. A naive approach is to use the *Merkle Hash Tree* (MHT) [8] as the ADS and apply the conventional MHT-based authentication methods. However, this approach has three major drawbacks. First, an MHT supports only the query keys on which the Merkle tree is built. To support queries involving an arbitrary set of attributes, an exponential number of MHTs need to be constructed for each block. Second, MHTs do not work with set-valued attributes. Third, MHTs of different blocks cannot be aggregated efficiently, making it incapable of leveraging inter-block optimization techniques.

To overcome the aforementioned drawbacks, we proposed a novel ADS in our prior work [1], based on the cryptographic multiset accumulator [9]. The multiset accumulator supports the following operations:

- Given a multiset  $X$ , it can compute an accumulative value  $acc(X)$ , which is a collision-resistant digest of the multiset.
- Given two multisets  $X_1, X_2$  where  $X_1 \cap X_2 = \emptyset$ , a disjoint proof  $\pi$  can be computed.
- Given two accumulative values  $acc(X_1), acc(X_2)$ , and the disjoint proof  $\pi$ , one can verify that  $X_1 \cap X_2 = \emptyset$ .
- Given two accumulative values  $acc(X_1)$  and  $acc(X_2)$ , the accumulative value of the multiset  $X_1 + X_2$  can be computed as  $acc(X_1 + X_2) = acc(X_1) + acc(X_2)$ .

To compute the digest of an object  $o_i = \langle t_i, V_i, W_i \rangle$ , we first transform the numerical vector  $V_i$  into a set of binary prefix attributes. For example, a value 4 is transformed as  $trans(4) = \{1*, 10*, 100\}$ , where  $*$  denotes the wildcard matching operator. Then, the digest is computed as the accumulative value

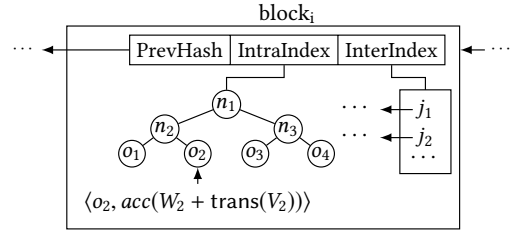


Figure 2: Intra-block and Inter-block Indexes

of both the set-valued attribute  $W_i$  and the transformed numerical vector  $trans(V_i)$ , i.e.,  $Digest_i = acc(W_i + trans(V_i))$ . This digest is used to serve as the ADS.

Given a data object and a query condition, there are only two possible outcomes: match or mismatch. The soundness of the first case can be easily verified by returning the object directly, since its integrity can be authenticated by its hash stored in the block header. For the more challenging second case, the object's digest is used. Observe that a Boolean function expressed in conjunctive normal form can be viewed as a list of sets. For example, a query condition "Sedan"  $\wedge$  ("Benz"  $\vee$  "BMW") is equivalent to two sets: {"Sedan"} and {"Benz", "BMW"}. Consider a mismatching object {"Van", "Benz"}. It is easy to observe that there exists an equivalence set (i.e., {"Sedan"}) such that its intersection with the object's attribute is empty. As such, we can generate a *disjoint proof*  $\pi$  of {"Sedan"} and {"Van", "Benz"} to attest the mismatch without returning the original object. As for range query condition, it is first transformed into an equivalent Boolean function over its binary prefixes. For example, a query range  $[0, 6]$  is equivalent to the Boolean function  $0 * \vee 10 * \vee 110$ . After transformation, a range query can be processed in the same manner as a Boolean query. Interested readers are referred to [1] for more details.

### 2.2 Batch Verification

For each query, there can be many objects lying in the query's time period  $[t_s, t_e]$ . Although one can process those objects one by one, a better approach is to process them in batch based on the query conditions. To this end, three batch verification techniques are developed in vChain.

**Intra-block Index.** The idea is based on the following observation. If two objects share some common attribute value, they may mismatch a query due to the same query condition. As such, to reduce the proving and verification overheads, vChain builds an intra-block index that aggregates the objects within a block. The index is an MHT-like binary tree built in a bottom-up fashion. Specifically, for each object, a leaf node is created. Then, it recursively merges two most similar tree nodes until the root is obtained. For each non-leaf node  $n$ , an attribute set is computed as the union of those of the child nodes, i.e.,  $W_n = W_{n_l} \cup W_{n_r}$ , where

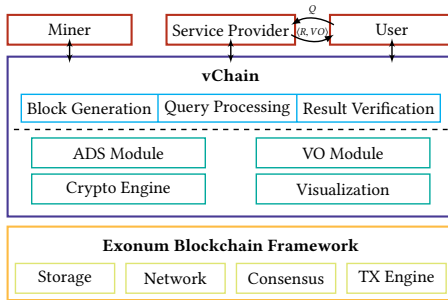


Figure 3: System Architecture of vChain

$n_l$  and  $n_r$  be the left and right children of node  $n$ , respectively. We also compute its digest as  $\text{acc}(W_n)$  and its hash value as  $\text{hash}(\text{hash}(\text{hash}_{n_l}|\text{hash}_{n_r})|\text{Digest}_n)$ , where  $\text{hash}(\cdot)$  is a cryptographic hash function and  $|$  is the string concatenation operator. With the intra-block index, a tree node and its corresponding digest can be used to prove that all the underlying objects mismatch the same query condition, thereby improving the query performance.

**Inter-block Index.** The inter-block index works in a fashion similar to the intra-block index. The difference is that it is used to aggregate the objects across multiple blocks. It is based on the skip list, where each jump consists of a hash of the skipped blocks, an attribute set that aggregates all objects in the jump, and a corresponding accumulative value. The inter-block index can help to boost the query performance when all objects in a jump mismatch the same query condition. The intra- and inter-block indexes are illustrated in Fig. 2.

**Online Batch Verification.** The indexes attempt to cluster the objects of the same block or across blocks in a way to maximize the proving efficiency of mismatching objects. Nevertheless, some objects indexed in different blocks or even different subtrees of the same block may also share the same reason of mismatching. Thanks to the additive homomorphic property of the multiset accumulator [9], vChain further aggregates such objects online for greater efficiency.

### 3 SYSTEM OVERVIEW

We implement the vChain prototype system based on the open-source Exonum framework (version 0.13-rc2)<sup>1</sup> using Rust programming language. Figure 3 shows the system architecture, which consists of three layers.

The bottom layer is the Exonum Blockchain framework. It offers the essential functionalities of data storage, P2P networking, blockchain consensus protocols, and blockchain transaction executions.



The middle layer of the vChain system consists of several low-level modules. The cryptographic engine provides the functions of computing cryptographic hash function, set

accumulative values, and set disjoint proofs. The vChain system uses 256-bit BLAKE2b as the hash function. For set accumulative values and their corresponding set disjoint proofs, vChain uses pairing over the BLS12-381 curve, which is implemented in the ZEXE library [10]. Furthermore, vChain uses Rayon<sup>2</sup> for data parallelism to accelerate the computation. The ADS module and the VO module are responsible for computing the ADS and VO, respectively, as introduced in Section 2. Finally, the visualization module is implemented to visualize the data structures of our proposed ADS and VO and provide a better demonstration experience.

The top layer of the vChain system offers the core functions of the verifiable Boolean range queries, including: (i) ADS generation for the miners, (ii) query processing for the SP, and (iii) result verification for the users.

### 4 DEMONSTRATION DETAILS

In our demonstration, we showcase how the vChain system can support verifiable query processing in the following scenarios.

**Scenario 1: Walk-Through of Query Life Cycle.** A lightweight user can issue a Boolean range query on Panel 1 of Fig. 4 by entering the time window (block IDs), the numerical range, and the Boolean function. After that, the query results and VO will be shown on Panel 2 and Panel 3, respectively. A  mark on the result panel indicates that the result verification succeeds while a  mark reveals the unsoundness or incompleteness of the results. The verification time and VO size are displayed on Panel 2 to show the performance. The user can further click the VO on Panel 3 to view the detailed information of the corresponding VO on Panel 4, which visualizes the VO structure with the returned objects.

**Scenario 2: Simulation of Attack Detection.** For better insights, we demonstrate how a user can detect attacks. Given a query, attacks can be divided into two categories: (i) *Unsound*: Some object in the results mismatches the query condition or has been tampered with. (ii) *Incomplete*: Some object that matches the query condition is missing from the results. We can simulate such attacks by clicking the “Attack” button in the SP dashboard (Fig. 5). For example, assume the correct results for a query are  $\{o_{50}, o_{51}\}$ . A compromised SP may conduct attacks by (i) tampering with the content of the object  $o_{50}$ , and (ii) not returning the object  $o_{51}$  but generating a set disjoint proof for  $o_{51}$ . Once receiving the results, the user will get the following error messages as Fig. 6 shows: ① the returned object  $o_{50}$  does not match the hash digest, which invalidates the soundness; ② the set disjoint proof for  $N_{51}$  is incorrect, indicating some object is missing from the results.

<sup>1</sup><https://exonum.com/>

<sup>2</sup><https://github.com/rayon-rs/rayon>

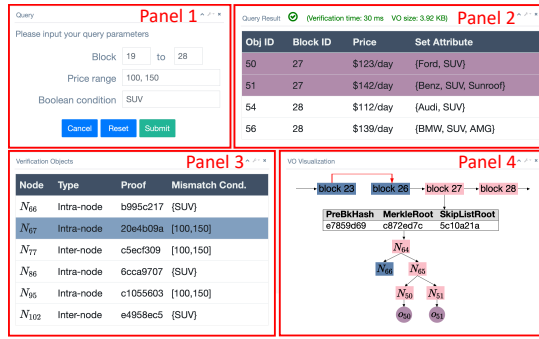


Figure 4: User Dashboard

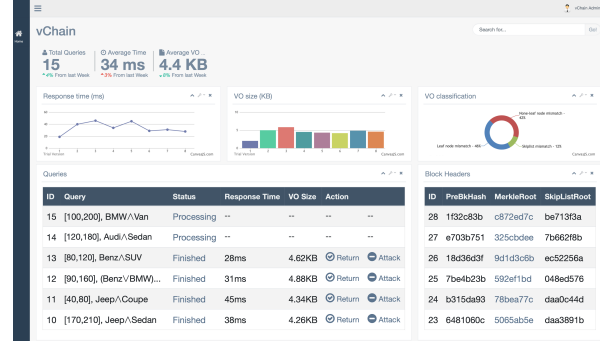


Figure 5: SP Dashboard

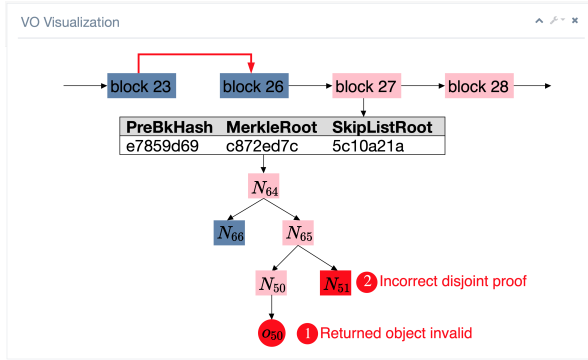


Figure 6: Attack Detection

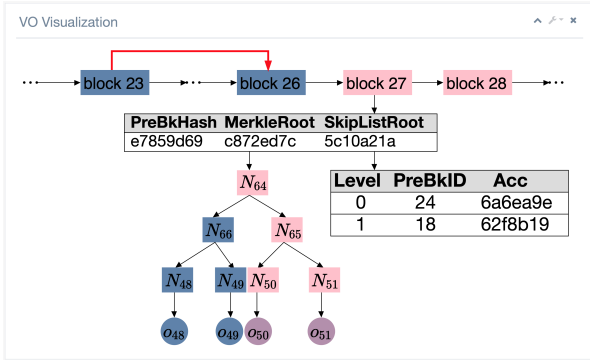


Figure 7: SP ADS Inspection

**Scenario 3: Understanding Batch Verification Performance.** We also demonstrate the effectiveness of the indexed batch verification techniques. To do so, we can instruct the SP to choose different system configurations (intra-block index, inter-block index, or both) and compare the query performance in terms of query processing time, VO size, and result verification time. Statistics such as total query count and average query processing time will be shown in the SP dashboard (Fig. 5). To inspect the detailed information of a completed query, we can click the status of the query to show its results together with the visualized ADS and VO structures (see Fig. 7).

## ACKNOWLEDGMENTS

This work is supported by Research Grants Council of Hong Kong under GRF Projects 12201018 & 12200819 and CRF Project C1008-16G.

## REFERENCES

- [1] C. Xu, C. Zhang, and J. Xu. 2019. vChain: Enabling verifiable boolean range queries over blockchain databases. In *Proc. SIGMOD*, 141–158.
- [2] A. Sharma, F. M. Schuhknecht, D. Agrawal, and J. Dittich. 2019. Blurring the lines between blockchains and database systems: the case of hyperledger fabric. In *Proc. SIGMOD*, 105–122.
- [3] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. 2019. Towards scaling blockchain systems via sharding. In *Proc. SIGMOD*, 123–140.
- [4] C. Zhang, C. Xu, J. Xu, Y. Tang, and B. Choi. 2019. GEM<sup>2</sup>-Tree: A gas-efficient structure for authenticated range queries in blockchain. In *Proc. ICDE*, 842–853.
- [5] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang. 2019. Fine-grained, secure and efficient data provenance on blockchain systems. *Proc. VLDB Endow.*, 12, 9, 975–988.
- [6] Blockchair. 2018. A blockchain search and analytics engine for Bitcoin, Bitcoin Cash and Ethereum. (2018). <https://blockchair.com/>.
- [7] BigchainDB GmbH. 2018. BigChainDB 2.0: The blockchain database. (2018). <https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf>.
- [8] R. C. Merkle. 1989. A certified digital signature. In *Advances in Cryptology — CRYPTO*, 218–238.
- [9] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei. 2018. Authenticating aggregate queries over set-valued data with confidentiality. *IEEE Trans. Knowl. Data Eng.*, 30, 4, 630–644.
- [10] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. 2018. Zexe: Enabling decentralized private computation. Cryptology ePrint Archive, Report 2018/962. (2018).