# Vassago: Efficient and Authenticated Provenance Query on Multiple Blockchains

Rui Han*, Jiang Xiao*, Xiaohai Dai*, Shijie Zhang*, Yi Sun†, Baochun Li§, 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, China
†University of Chinese Academy of Sciences, China
§University of Toronto, Canada
jiangxiao@hust.edu.cn

***Abstract*—Recent successful pilot studies on blockchains showed significant benefits of data provenance towards improved visibility and authenticity, as well as a reduction of administrative costs. Traditionally, all the parties reside in a single blockchain system, which leads to various single-chain provenance query schemes. However, for the sake of convenience, it has been a trend for different parties to deploy their own blockchain systems separately, which requires cross-chain provenance queries. Unfortunately, state-of-the-art blockchain query schemes suffered from inauthentic transactions and low efficiencies when expanding to adversarial cross-chain commercial deals. To be more specific, query results may be inconsistent and incomplete over multiple blockchains due to the lack of global knowledge on cross-chain dependencies. Moreover, these schemes perform cross-chain provenance queries in sequence, leading to high query latencies. In this paper, we present Vassago, the first multi-chain system that empowers efficient and authenticated provenance queries. Vassago incorporates three innovative designs: 1) it explores the dependencies of cross-chain transactions, 2) it validates the authenticity of cross-chain provenance by recording and querying the dependencies on a shared blockchain, and 3) it improves efficiency by parallelizing query processes. Our experimental results show that Vassago can shorten the query latency by $85.9\%$ and reduce the storage consumption by up to $85.7\%$, with negligible overhead.**

***Keywords*-Blockchain, Cross-chain, Data provenance, Traceability**

## I. Introduction

Due to their merits of immutability and transparency, blockchain systems have been deployed in many fields, such as industry supply chains. However, in many scenarios, not all the parties are able to participate in a single blockchain system. For example, in a supply chain for COVID-19 vaccines, suppliers from different countries may need to cooperate to establish a supply chain over a blockchain system. However, due to the lack of scalability of a single blockchain and various regulatory policies from different countries, it is impractical to record the vaccine transactions from all the relevant suppliers in a single blockchain. Instead, multiple blockchains must be utilized, each of which is deployed in its own country. As a result, cross-chain queries are required to trace the circulation of vaccines across these blockchains.

Compared with single-chain queries, cross-chain queries are presented with two major challenges: lack of guarantees for data authenticity and longer query latencies. To be more specific, the consensus in a single chain can ensure the consistency of query results. In contrast, there is no such cross-chain consensus covering multiple chains. Consequently, results from cross-chain queries may not be authentic, as they may be either inconsistent or incomplete. As shown in Figure 2, query result from blockchain 2 and 3 must be consistent, and a complete historical record should be obtained from blockchain 1. Besides, a cross-chain query consists of multiple sub-query tasks on each blockchain, each of which will incur additional query latencies.

Although there are some cross-chain technologies, they can only support the interaction of two blockchains and guarantee the consistency between a pair of transactions. However, a cross-chain query usually involves the interactions across more than two blockchains, which are not able to utilize existing cross-chain technologies directly. Take a supply chain consisting of three blockchains (A, B, and C) as an example. Although the cross-chain technology can ensure the consistency between chain A and B or between B and C, it cannot deal with the consistency between chain A and C. Furthermore, existing cross-chain technologies mainly focus on the execution logic, which ignores the implementation of query functions.

To deal with the challenges above, we exploit the relationship among cross-chain transactions and successfully find the cross-chain provenance dependency. This dependency can indicate if a transaction in one chain has a corresponding transaction in the other chains. With this dependency, the consistency and completeness of the query results can be verified, thus guaranteeing the data authenticity of cross-chain queries. Besides, once acquiring the dependency, the sub-query tasks in a cross-chain query can be executed in parallel, thereby accelerating the overall efficiency.

Based on the cross-chain provenance dependency, we propose an efficient and authenticated query mechanism for the cross-chain scenarios, Vassago, which contains an adaptive two-layer architecture. The first layer contains a *Dependency Blockchain* (DB), which records the cross-

chain provenance dependency. The second layer includes a *Transaction Blockchain* (TB) Group, which contains a group of TB executing intra-chain and inter-chain transactions. Users will join the TBs on demand which is defined in the DB.

To evaluate Vassago, we have implemented a prototype system on top of Hyperledger Fabric [1] and conduct multiple experiments on our prototype implementation. Our experimental results demonstrate Vassago's high efficiency in executing cross-chain queries. Compared with the system without dependency, Vassago can reduce the latency of a data query by 85.9%. Our experimental results also show that Vassago is scalable. It can reduce storage overhead by up to 85.7% while providing comparable *transactions per second*.

Our original contributions in this paper are as follows:

- We present an in-depth analysis of the fundamental challenges in cross-chain scenarios: a lack of guarantees for data authenticity and longer query latencies.
- We identify the cross-chain provenance dependency, based on which we propose Vassago to support authenticated cross-chain data queries efficiently.
- We demonstrate the efficiency and scalability of Vassago with an extensive array of experiments. Our experimental results show that Vassago improves the efficiency of cross-chain queries, reduces unnecessary storage overhead, and provides a comparable throughput as the state-of-the-art.

The remainder of this paper is organized as follows. In Section II, we present the limitations of the current cross-chain techniques and describe the problem caused by a lack of the cross-chain provenance dependency. In Section III, we propose Vassago design principles and system architecture. We clearly define the cross-chain provenance and elaborate the cross-chain data query process supported by Vassago in Section IV. Our theoretical analysis of Vassago's performance is presented in Section V. The feasibility and scalability of Vassago are evaluated by our experiments in Section VI. Finally, we briefly present the related work, along with further discussions and some concluding remarks in Section VII, VIII, and IX, respectively.

## II. Background and Motivation

Current cross-chain technologies focus on cross-chain correctness in the transaction execution, but they could not satisfy functional requirements. In this section, we consider the provenance query requirement and figure out why the current cross-chain transaction could not get on well with the requirement.

### A. Limitations of the Cross-chain Strategies

Side-chain [2]–[4] and relay-chain [5]–[7] are the mainstream cross-chain techniques. Side-chain enables the interoperability of two different blockchains, in which the side-

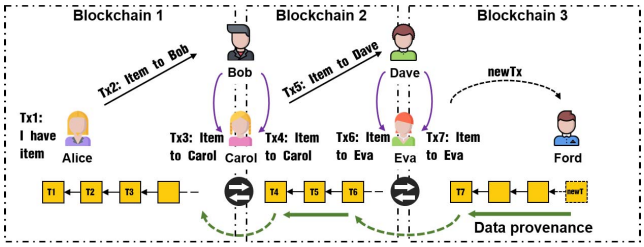



Figure 1. Cross-chain Data Provenance Scenario

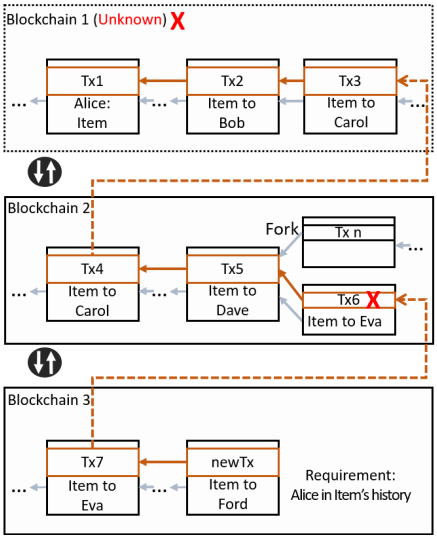



Figure 2. Overall Workflow of Cross-chain Data Provenance in Ford's view

chain acts as an extension of the main blockchain, and can validate data from the main blockchain through the *Simplified Payment Verification* (SPV) proof. Relay-chain alternatively adopts an intermediate blockchain to record cross-chain transactions on all interactive blockchains, which guarantees the existence of the transaction. Both succeed in ensuring the atomicity of transaction execution among different blockchains (e.g., atomic swaps). However, side-chain and relay-chain could not fulfill all the data provenance requirements separately. The limitations that side-chain and relay-chain suffer from are concluded as follows.

- **Limitation 1: low efficiency.** In a data provenance query, the query target in upstream is decided by the downstream query result, and the query will serially iterate each block. In the multi-chain scenario, we can perform queries in multiple blockchains at the same time. However, due to the lack of a trusted dependency for the target in different blockchains, the query is still carried out in the order from the downstream blockchain to the upstream blockchain, which leads to low query efficiency.
- **Limitation 2: authenticated requirement.** A dependency in a group of cross-chain transactions could

not be explicitly recorded and be consented to by all participators. As a result, the dependency is unauthenticated for the data provenance query. This can lead to two problems in the cross-chain provenance query, namely inconsistency and incompleteness. **Limitation 2.1: inconsistency.** Since a cross-chain transaction is a group of transactions recorded in different blockchains and current cross-chain strategies lack the ability to perceive the change of history records, this query for these cross-chain transactions cannot be verified as consistent. **Limitation 2.2: incompleteness.** Since the cross-chain provenance dependency could not be credibly recorded in different blockchains, it hurdles the query executor to determine whether the transaction is a cross-chain transaction, which leads to incomplete query results.

### B. A Motivating Example

We use a scenario in Figure 1 to describe the problem lacking the cross-chain provenance dependency in the cross-chain data provenance.

We first consider a pipeline supply-chain scenario with data provenance requirements. When all transactions happened in one blockchain (i.e., *Blockchain 1*), the downstream transaction could record the dependency with its upstream transaction. For example, *Tx3* records its upstream dependency with *Tx2*, so the item's tracking can perform one by one.

Now, consider a scenario with three blockchains (i.e., *Blockchain 1, Blockchain 2, Blockchain 3*). It is worth noting that there is no interoperation between *Blockchain 1* and *Blockchain 3* so they are unaware of each other's existence. When Ford wants to track the item history and then decides whether to make a transaction with Eva according to the query result as Figure 2, he will face the following challenges:

First, the query is inefficient **(Limitation 1)**. Only after tracking in *Blockchain 3*, can we know *Tx7* has a cross-chain transaction dependency with *Tx6* in *Blockchain 2*. Similarly, after tracking in *Blockchain 2*, we know *Tx4* has a cross-chain transaction dependency with *Tx3* in *Blockchain 1*. Hence the tracing is serial. If we can perceive the complete dependencies before the query, we can track the provenance in all the three related blockchains in parallel. More importantly, the cross-chain provenance dependencies cannot be agreed upon by different blockchains, so the dependencies are not trusted, leading to the following problem.

Second, the query may be inconsistent **(Limitation 2.1)**. There is a cross-chain transaction dependency between *Tx6* in *Blockchain 2* and *Tx7* in *Blockchain 3*. When *Blockchain2* forks, *Tx6* will be erased, making the transaction inconsistent with the other blockchains, resulting in an incorrectness problem. The problem will not happen if all the transactions are in one blockchain since the dependency can be consented

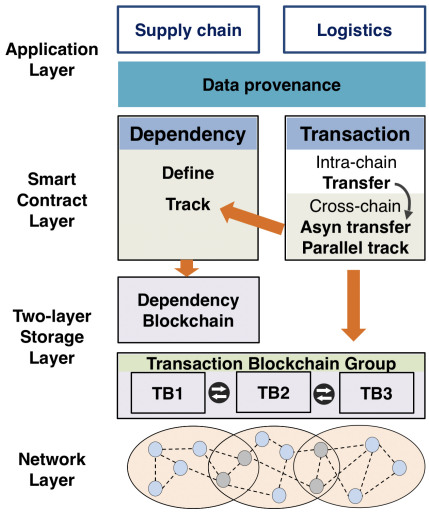



Figure 3. Vassago Architecture

to by all nodes with the same result. In other words, if the dependency about cross-chain transactions is consented to by all participators, the inconsistent record will be perceived.

Third, the query may be incomplete **(Limitation 2.2)**. We consider a scenario that Ford does not perceive the existence of *Blockchain 1*, and he does not know there is a cross-chain transaction dependency between *Tx4* and *Tx3*, so he could miss the provenance in *Blockchain1*. If all blockchains' existence and the cross-chain provenance dependencies can be integrated and consented by all nodes, any sub blockchain will realize the existence of the other blockchains without repudiation.

## III. System Design

Vassago is designed to avoid the limitations described in Section II so as to enable *efficient* and *authenticated* cross-chain queries. In this section, we will describe the design principles and elaborate on Vassago architecture based on these design principles.

### A. Design Principle

The entire design of Vassago is based on two principles: (1) cross-chain transactions are recorded without being tampered with; (2) query results are consistent with what is recorded before. Detailed descriptions of the design principles are as follows:

- *Avoid the cross-chain transactions to be tampered with in the Byzantine environment.* To make the cross-chain transactions authentic for the data provenance query, it should be agreed upon by all the cross-chain participators. Therefore, we should use a blockchain maintained by cross-chain participators to record the existence of the cross-chain transactions.
- *Guarantee the cross-chain transactions trusted for the data provenance query.* The key to enabling trustworthy relevance is that each node's cross-chain behavior

should be consistent with each other. As a result, we design the corresponding smart contracts to record the relevance of cross-chain transactions.

### B. Vassago Architecture

As shown in Figure 3, the overall design of Vassago specifies a system architecture consisting of four layers: the application layer, the smart contract layer, the two-layer storage layer, and the network layer.

The application layer provides a provenance track in pipe-like information transfer scenarios like supply chain and logistics.

In the smart contract layer, it provides interfaces to interact with the underlying blockchain. There are two types of API. The first is dependency API, which is used to update and query cross-chain dependency described in Section IV-A. The second is transaction API, containing parallel cross-chain data query and transaction execution. When the transaction is a cross-chain transaction, it will upgrade to an asynchronously cross-chain transaction. The cross-chain read and write operation calls for dependency read API because they should follow the cross-chain dependency described in Section IV-A.

In the storage layer, we design a two-layer storage structure. The first layer is the *Dependency Blockchain* (DB), which records each item's cross-chain dependency graph and can be called by dependency API. The second layer is a group of the *Transaction Blockchains* (TBs). TBs perform as traditional blockchain to record inter-and intra-chain transactions. They will execute operations by calling dependency API. It is worth noting that a TB can dynamically attach to the system so that the cross-chain provenance dependency may change in the running process.

Finally, we consider how nodes join the DB and TBs according to the cross-chain provenance dependency in the network layer.

## IV. Data Query by Vassago

This section dwells on the cross-chain data query mechanism supported by Vassago. We first present the definition and property of the cross-chain provenance dependency, and generate a dependency graph to guide the cross-chain data query. Then, we introduce the execution process of cross-chain data query that guarantees both authenticity and efficiency.

### A. Dependency Definition

Specifically, the cross-chain provenance dependency depicts the dependencies between different cross-chain transactions, by which we can infer the item transfer rules between multiple blockchains. The provenance dependency can be reused for similar transactions which have the same transaction footprint among multiple blockchains. Therefore, compared with a transaction, a provenance dependency is more static for operation and has fewer states in terms of storage.

The smallest unit of cross-chain provenance dependency is the cross-chain transaction dependency. A cross-chain transaction dependency contains a group of transactions that happened in different blockchains with implicit dependency. In other words, they should be executed atomically. A dependency should contain at least two transactions in different blockchains because they are divided into two groups called upstream and downstream. The rule is that transactions in the downstream group should happen after those in the upstream. If and only if the number of one of the upstream or the downstream is bigger than 2, the cross-chain transaction is defined as a multi-dependent transaction. Otherwise, it is a single-dependent transaction. The math definition of *cross-chain transaction dependency* is described as following.

*Definition 1: Cross-chain transaction dependency. We denote a transaction as $Tx$, the upstream of $Tx$ as $\rho(Tx)$, the downstream of $Tx$ as $\omega(Tx)$, the belonging blockchain of $Tx$ as $B(Tx)$. Giving a group of transactions $Tx = \{Tx1, Tx2, ..., Txn\}(n \geq 2)$, for any transaction $Txi \in Tx$, we define $Txi' = Tx - \{Txi\}$. A cross-chain transaction dependency $Rcro$ satisfies the following properties:*

$$(\omega(Txi) \cap \rho(Txi') \neq \varnothing \vee \rho(Txi) \cap \omega(Txi') \neq \varnothing) \wedge B(Txi) \cap B(Txi') = \varnothing \quad (1)$$

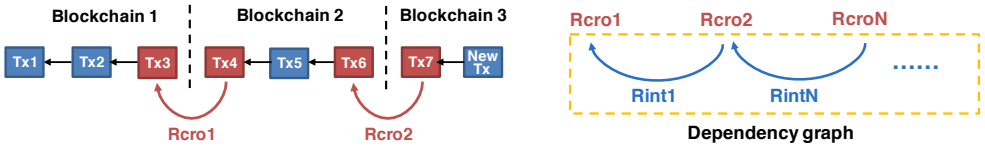
The *cross-chain provenance dependency* reveals the relations between multiple *cross-chain transaction dependencies*. Similar to *cross-chain transaction dependency*, a *cross-chain provenance dependency* contains a group of *cross-chain transaction dependencies*. The math definition of *cross-chain provenance dependency* is described as following.

*Definition 2: Cross-chain provenance dependency. We denote a group of cross-chain provenance dependency as $Rcro = \{Rcro1, Rcro2, ..., Rcron\}(n \geq 2)$. For any dependency $Rcroi \in Rcro$, we define $Rcroi' = Rcro - \{Rcroi\}$. An cross-chain provenance dependency $Rint$ satisfies the following properties:*

$$\omega(Rcroi) \cap \rho(Rcroi') \neq \varnothing \vee \rho(Rcroi) \cap \omega(Rcroi') \neq \varnothing \quad (2)$$

Finally, to guide cross-chain transaction execution and query, we need to link all the *cross-chain provenance dependencies* into a route as a description for an item's complete dependency among blockchains.

*Definition 3: Dependency graph. Given a group of cross-chain provenance dependency as $Rint = \{Rint1, Rint2, ..., Rintn\}$ for one item. The cross-chain transaction route is a directed graph $G = (\tau, \varepsilon)$,*




(a) Cross-chain Transaction Dependency (b) Cross-chain Provenance Dependency

Figure 4. Cross-chain Dependency Graph

*which satisfies the following properties:*

$$\begin{aligned} &\tau = Rint \\ &\varepsilon = \{(Rinti, Rintj) | \rho(Rinti) \cap \omega(Rintj) \neq \varnothing\} \\ &\quad where \quad Rinti, Rintj \in Rint. \end{aligned} \tag{3}$$

Figure 4 depicts the *cross-chain transaction dependency* and the *cross-chain provenance dependency* in the case of the single-dependent transaction. Taking the scenario presented in Section II as an example again, the item first comes from *Blockchain 1*, and then it is developed by *Blockchain 2* and finally distributed in *Blockchain 3*. Each group contains multiple enterprises. The transaction in a group that is not dependent on other groups' transactions is denoted by the blue square, while the transaction that is dependent on other groups' transactions (i.e., cross-chain transaction) is denoted by the red square. A black arrow denotes a transaction dependency that happens in one group, and a red arrow denotes a *cross-chain transaction dependency*. There are two *cross-chain transaction dependencies* shown in this figure: the dependency $Rcro1$ between *Tx3* and *Tx4* and the dependency $Rcro2$ between *Tx6* and *Tx7*. The dependency direction is from the downstream enterprise to the upstream enterprise. A blue arrow denotes the *cross-chain provenance dependency*, for instance, the provenance dependency $Rint1$ links a downstream transaction dependency $Rcro2$ with an upstream transaction dependency $Rcro1$ in this figure. Besides, some other *cross-chain provenance dependencies* $RintN$ initiated from *Blockchain N* also exist. The set of all the blue arrows will constitute a complete provenance dependency graph for one item.

### B. Adaptive Two-layer Blockchain

In the four-layer architecture, the two-layer storage plays a vital role. Next, we elaborate on the two-layer storage containing the *Dependency Blockchain* (DB) and the *Transaction Blockchain* (TB).

As mentioned before, every node can choose to join any blockchain in Vassago on demand. There are two roles that a participator can play to join a blockchain, called consensus node and witness node. A consensus will take part in the blockchain consensus, which will cost the devices computing power, and more consensus nodes lead to more consensus difficulty. A witness node will record and verify the results passed by the consensus node without a consensus process. A witness node will not increase consensus difficulty. Participators could join the blockchain as a consensus node or a witness node depending on their demands.

Some specific examples for describing the join rule are shown in Figure 5(a). As a cross-chain dealer like Carol, she first joins the DB as a consensus node to define the cross-chain dependency, then she joins the upstream TB and the downstream TB as a consensus node to execute the cross-chain transaction.

As a cross-chain tracer like Ford, he joins the DB as a witness node to read the cross-chain provenance dependency, then enters the related blockchain TB1, TB2 based on the dependency defined in DB as a witnesses node to trace the data provenance in all corresponding blockchains.

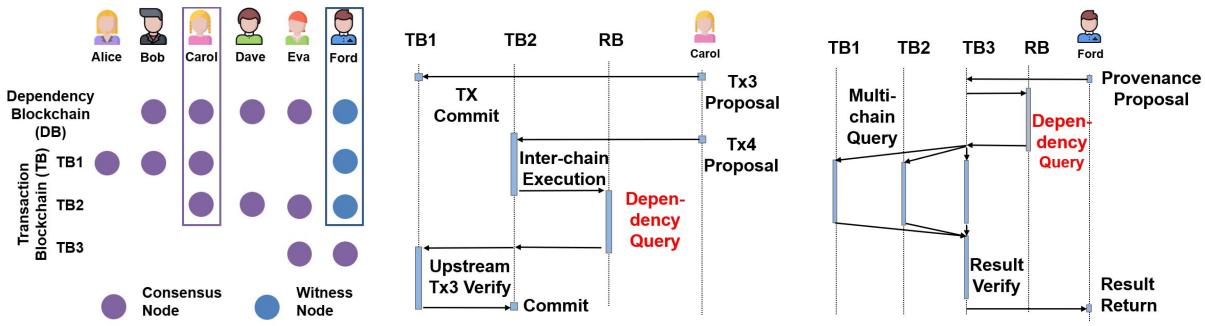
Usually, some nodes do not care about the cross-chain behaviors like Alice, so they do not take part in the DB and only join the TB where she originally belongs to.

### C. Authentic Query by Dependency

We show how Vassago enables authentic cross-chain queries in this subsection. In general, Vassago makes use of the cross-chain provenance dependency recorded in DB to ensure the authenticity of query results. We implement the cross-chain data query on the smart contract. The user can obtain the historical transaction records through the smart contract and judge whether the transaction is executed according to the cross-chain provenance dependency.

We take the cross-chain dealer (i.e., Carol) in Figure 5(a) as an example. She first reaches a consensus on the cross-chain provenance dependency with Bob in DB. There are two transactions *(i.e., Tx3, Tx4)* that record the item's belonging with her. We assume the new owner will execute the transaction to simplify the transaction, so Carol needs to complete *Tx3* and *Tx4* in different blockchain (i.e., TB1, TB2). It is worth noting that *Tx3* and *Tx4* are a couple of cross-chain transactions because there is a cross-chain transaction dependency that indicates *Tx4*'s dependency on *Tx3*.

The cross-chain transaction execution process is shown in Figure 5(b). *Tx3* can be directly executed in TB1 with Bob's consensus because *Tx3* could find its dependent transaction *Tx2* in the same blockchain. When Carol wants to make a transaction with Dave, she needs first to record *Tx4* for Dave to prove her ownership of the item by calling TB smart contract in TB2. The process of *Tx4* execution is as follows.




(a) Adaptive Two-layer Blockchain (b) Cross-chain Transaction Execution (c) Cross-chain Data Provenance Query

Figure 5. Processing of Vassago: Transaction Execution and Query

Once a cross-chain transaction is proposed (*Tx4* in TB3), Vassago first examines whether its cross-chain dependency exists in DB (Dep1). If it exists, Vassago finds the dependent blockchain and validates whether the dependent transaction has been executed in this chain (*Tx3* in TB2). If so, the newly proposed cross-chain transaction can be safely executed. It is worth noting that the dependency could be reused for many items if they have the same cross-chain provenance dependency with this item.

In terms of the provenance query, the query executor needs to verify whether the provenance query result matches the cross-chain provenance dependency defined in the DB. We present two main properties of the authenticity of the cross-chain provenance query:

**Soundness.** Soundness means inconsistent cross-chain transactions are appreciable in the query process. The tamper-proof property is guaranteed by DB because it stores the cross-chain provenance dependency, and cross-chain operations will follow the dependency. Since the objective of Vassago is to enable multi-chain queries, whose query results correspond to the exiting data status, if there is a blockchain fork before query processes, the query results will be inconsistent. At this time, the inquirer can use the provenance dependency to judge the wrong result. We can avoid **limitation 2.1** by this verification strategy.

**Completeness.** Completeness means the inquirer should visit all blockchains which relate to the target item's history. Because the broken transaction dependency causing by the cross-chain scenario is renovated by the dependency graph, which links all the cross-chain provenance dependencies into an integrity route, the inquirer could perceive the global view of an item's history among different blockchains. Therefore he will not miss any blockchains that properly have the item history records. According to this, we can avoid **limitation 2.2**.

### D. Efficient Query by Parallelization

Vassago improves the query efficiency by parallelizing the query tasks on multiple chains based on the cross-chain provenance dependency recorded on DB.

Figure 5(c) depicts the process of the cross-chain data provenance in Ford's view. The cross-chain data provenance query consists of five steps: item trace proposal, dependency query, cross-chain query, result verification, and result return. To be more specific, if Ford wants to make a query among multiple blockchains, he needs to first propose the item trace proposal to the smart contract in TB3. As a response, the smart contract queries the DB for the cross-chain provenance dependency related to the target item. Afterward, TB3 can execute the trace on all related TBs according to the item's cross-chain provenance dependency. Since the item's global dependency among different blockchains has been consented to and recorded by all cross-chain participators, the data provenance query process can be executed in parallel to avoid **limitation 1**. The query results will be verified to ensure their authenticity. If the results pass the verification, they will be returned to Ford finally.

## V. Theoretical Analysis

Vassago incurs little additional overhead to cross-chain data provenance. It cuts off unnecessary storage, and the design itself will not bring additional compute or storage costs for the data provenance query. With the help of Vassago, the provenance query could be efficient. In this section, we quantify the optimization capabilities of the above solutions.

We evaluate Vassago performance in two transaction dependency modes. The first is the single-dependency, which indicates that the dependency between transactions is one by one. The second is multi-dependency. According to the cross-chain transaction dependency definition in Sec. IV-A, we set each group of cross-chain transactions to contain three transactions, which includes two transactions in the upstream group and one in the downstream group. In this case, TBs arrange like a binary tree that the earliest upstream TBs are the leaf nodes and the latest downstream TB is the root node. To simplify the analysis, we assume that all participators in the downstream TB are interested in the transactions in the upstream, so they participate in the

upstream TBs as witness nodes. We denote the group of participators node as an organization.

### A. Reducing Storage Overhead

The number of transactions $M$ is determined by the number of initial information $I$ and the times of information transfers $T$, i.e., $M = I \times T$. For one piece of information, each node will participate in the information transmission process only once. When nodes are distributed into several blockchains, the total amount of information $I$ remains unchanged. For each blockchain, let $K$ and $T'$ represent the number of nodes and the number of information transfers, so $T' = T/K$. Thus, the number of transactions in each blockchain is similar. We denote the number as $N$, and it is:

$$N = T' \times I = \frac{M}{K} \tag{4}$$

In a single-dependency scenario among multiple blockchains, downstream blockchains choose to store upstream blockchains' transactions for provenance query. With the cross-chain provenance dependency, nodes can precisely store the related blockchain. We denote the number of organizations as O. In Vassago, the average storage cost for each node denoted as $S^s_{Vassago}$ is:

$$S^s_{Vassago} = \frac{1 + 2 + ... + O}{O} \times N \tag{5}$$

Without the dependency, nodes need to store all blockchains to try their best to achieve data provenance completeness. Each node's cost denoted as $S^s_{common}$ is as follows, which is almost $2\times$ in Vassago.

$$S^s_{common} = O \times N \tag{6}$$

In the multi-dependency scenario, to quantify calculations and facilitate experiments, we assume the dependency is two. According to the cross-chain provenance dependency defined in Sec. IV-A the blockchain will be arranged as a binary tree. The earliest upstream blockchains are the leaf nodes whose heights are 1, and the latest downstream blockchain is the root node which the height is the same as tree depth. In the layer where the height is $h$, and the tree depth is $d$, the nodes' storage cost denoted as $f(h)$ is:

$$f(h) = \begin{cases} (2f(h-1)+1) \cdot N, & h > 1 \\ N, & h = 1 \end{cases} \Rightarrow f(h) = (2^h - 1) \cdot N \tag{7}$$

The total storage cost in Vassago denoted as $Total(Vassago)$ is:

$$\begin{aligned} Total(Vassago) &= \sum_{h=1}^{d} 2^{d-h} \cdot f(h) \\ &= N \cdot (2^d \cdot (d-1) + 1) \end{aligned} \tag{8}$$

The total number of organizations denoted as $TotalO$ is

$$TotalO = \sum_{h=0}^{d-1} 2^h \tag{9}$$

In this case, the average storage for each node cost in Vassago denoted as $S^m_{Vassago}$ is as follows, while the space complexity is $O(d)$.

$$\begin{aligned} S^m_{Vassago} &= \frac{Total(Vassago)}{TotalO} \\ &= \frac{2^d \cdot (d-1) + 1}{(2^d - 1)} \end{aligned} \tag{10}$$

At this time, the node storage cost in the common solution denoted as $S^m_{common}$ is as follows, where the space complexity is $O(2^d)$.

$$\begin{aligned} S^m_{common} &= TotalO \cdot N \\ &= (2^d - 1) \cdot N \end{aligned} \tag{11}$$

### B. Avoiding Additional Overhead

Our design will not bring obvious storage overhead for the dependency. The dependency is the abstract of relationship among blockchain for each item, while each related blockchain contains much more transactions to record the particular item transfer history. In common cases, the dependency is much smaller compared with provenance history. We assume an item traveling among different blockchains in single dependency for easy to formulate the problem, and it will visit each blockchain one time. We denote the corresponding blockchain number as $B$. Transaction number in each blockchain is $K_i (0 < i < B)$. We denote dependency record number as $D$, transaction number relating to the item as $T$. We have the following equations.

$$D = 2 \times (B - 1) - 1 \tag{12}$$

$$T = 2 \times B - 1 + \sum_{i=1}^{B} K_i \tag{13}$$

$$In\ common\ case,\ \sum_{i=1}^{B} K_i \gg D,\ so\ T \gg D.$$

Plus, the dependency is reusable, which further reduces the dependency storage overhead.

### C. Improving Efficiency

Since the cross-chain provenance dependencies are reliably recorded in Vassago, the data provenance in each TB can be executed in parallel. The total query latency can be regarded as the query time on a single TB, denoted as $T_{Vassago}$, the velocity for querying a transaction as $v$, and the time cost in DB query is $T_{DB}$. We can obtain:

$$T_{Vassago} = \frac{N}{v} + T_{DB} \tag{14}$$

The data provenance time for an item's completeness history without the dependency denoted as $T_{common}$ is:

$$T_{common} = \frac{N \cdot O}{v} \tag{15}$$

Because $T_{DB}$ is small enough, which is described in Sec. V-B and verified in Sec. VI-B. Compared with the common solution without the cross-chain provenance dependency, Vassago can reduce the time cost to $1/O$.

## VI. Performance Evaluation

In this section, we conduct extensive experiments to demonstrate Vassago's feasibility and efficiency. We first introduce the experimental setup and then evaluate Vassago from three aspects: query latency, system throughput, and storage overhead.

### A. Experimental Setup

We implement the prototypes on top of Hyperledger Fabric v2.2, which supports multiple channels for different blockchains. Since our main design relies on dependency, we implement two prototypes with and without dependency respectively. In this part of the experiments, each organization will issue 100,000 cross-chain transactions. Based on the smart contracts in Fabric, we implement the functions to issue cross-chain transactions and cross-chain query requests.

We run the orderer group using a typical Kafka orderer setup with 3 ZooKeeper nodes, 4 Kafka brokers, and 3 orderers. To simulate the supply chain scenario, each party can join multiple Fabric channels as an organization. In each organization, we deploy 4 peers and generate $100,000$ transactions of 100 items in parallel. The proposal rate of cross-chain transactions depends on the number of items and the time taken to execute the transactions of each item.

Our experiments are conducted on the Alibaba Cloud Compute Optimized ECS Instance with 8 vCPUs and 16 GB RAM, Intel Xeon Platinum 3.30 GHz, with Ubuntu 18.04 LTS as the operating system. The experiments are set up as the description in Sec. V. In each experiment, we increase the total number of transactions from $100,000$ to $800,000$ by increasing the number of organizations running on each VM. We use Smallbank benchmark in BLOCKBENCH [8] to test the system performance with different read-write sets.

### B. Query Latency

We compare the query latency of systems with and without the dependency and depict the results as shown in Figure 6. Since each participator in Vassago stores the dependency, the inquirer can perform a query in different TBs in parallel. Without the dependency, the query latency increases linearly as the number of blocks increases. By contrast, it keeps almost unchanged in Vassago. In particular, when $800,000$ blocks are packaged, the query latency in Vassago is only $58.41ms$, which outperforms $410.56ms$ without the dependency. Besides, the larger is the number of organizations involved in the system, the larger is the reduction of query latency. To sum up, Vassago can significantly improve the efficiency of cross-chain data queries.

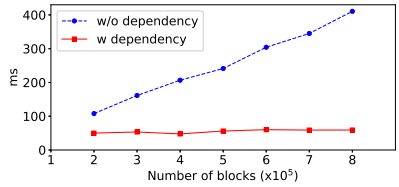



Figure 6. Query latency w and w/o the cross-chain dependency

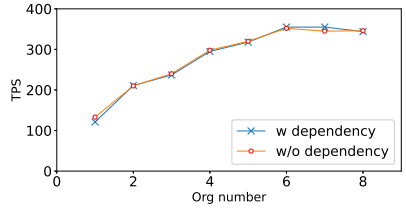


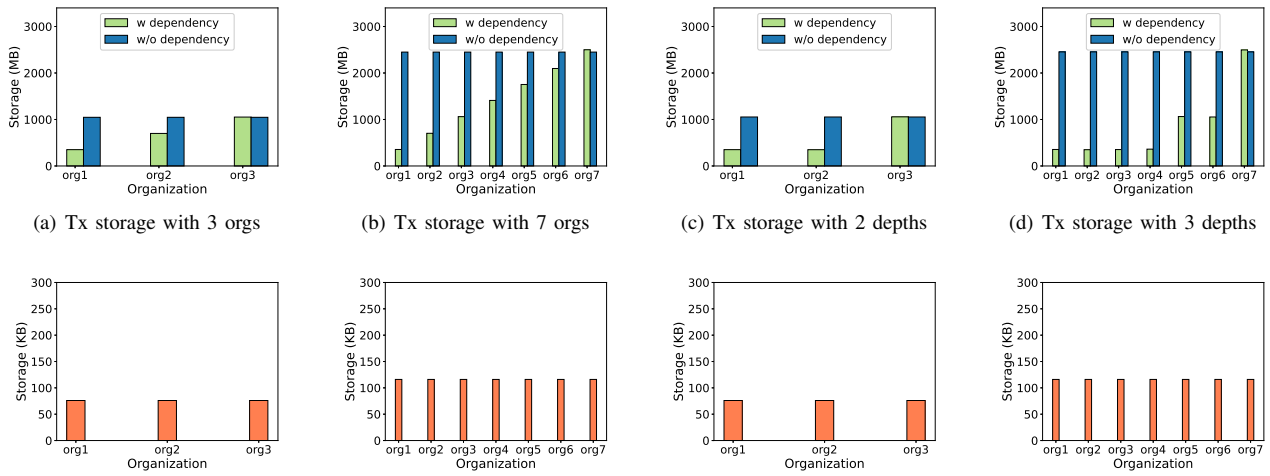

Figure 7. TPS w and w/o the cross-chain dependency

### C. System Throughput

To study if the cross-chain query technology will bring negative effects on the system performance, we compare the throughput between systems with and without dependency. The comparison results are shown in Figure 7. It can be easily found from the figure that whether the dependency is utilized or not, the system performance keeps similar to each other. To be more specific, the TPS gap between the two systems remains under 10 $tx/s$ if the number of transactions and nodes are set as the same. Particularly, when the number of nodes increases to a certain extent (e.g., the number of organizations is 6, and the number of nodes is 24), the system TPS with or without dependency peaks at around 352 $tx/s$ and 355 $tx/s$, respectively. Therefore, we can conclude that the dependency has negligible impacts on the overall system performance.

### D. Storage Overhead

To evaluate the storage overhead, we design two groups of experiments, each of which contains a different number of organizations/blockchains. Concretely speaking, one group has only 3 organizations (org1, org2, org3) participating in the data provenance. org1 and org2 are the upstream organizations, org3 is downstream of them. The other group contains 7 organizations org1-org7. The upstream organizations include org1, org2, org3, and org4. org5 and org6 are in




(a) Tx storage with 3 orgs (b) Tx storage with 7 orgs (c) Tx storage with 2 depths (d) Tx storage with 3 depths

(e) Dpd storage with 3 orgs (f) Dpd storage with 7 orgs (g) Dpd storage with 2 depths (h) Dpd storage with 3 depths

Figure 8. Comparison of the storage overhead of transaction (tx) and dependency (dpd), with different number of organizations (org) and different depths of dependency. (a), (b), (e), and (f) are evaluated with single-dependency, while (c), (d), (g), and (h) are evaluated with multiple-dependency.

the middle, while org7 is the downstream organization. We test each organization's storage overhead in these two groups with two kinds of dependency, namely single-dependency and multi-dependency. The experimental results are shown in Figure 8.

With regard to the experiments with single-dependency, based on the assumption we make in Sec. V, the largest storage overhead should be the downstream organization (i.e., org3 in Figure 8(a) and org7 in Figure 8(b)). As can be seen from the figures, the storage overhead in the downstream organization reaches 1052 MB and 2500 MB respectively, which is close to the solution without the dependency. The reason for it is that, without cross-chain dependency, participators need to store the data of all the blockchains to guarantee the result completeness. Besides, each organization will store a small number of the cross-chain dependency recorded in the TB. However, as shown in Figure 8(e) and Figure 8(f), the storage overhead caused by the dependency in TB accounts for less than 1‰ of the total storage.

As for the experiments with multi-dependency, we consider the same condition as assumed before. When the depth is 2 as Figure 8(c), org3 has the largest storage overhead, which is 3× higher than that of org1 or org2. However, it is also close to the solution without the dependency. When the depth is 3 in Figure 8(d), we can draw a similar conclusion that Vassago brings a similar overhead in the downstream party and less overhead in the middle/upstream parties. Besides, as shown in Figure 8(g) and Figure 8(h), the dependency storage overhead is 76 KB and 116 KB when the depth is 2 and 3 respectively, which also demonstrates the lightweight of the dependency.

To sum up, with the aid of dependency, Vassago reduces the storage overhead of the upstream and middle parties largely, and keeps the similar storage overhead to that without dependency.

## VII. Related Work

Blockchain-based data query has generated increasing interest in different applications. This section briefly presents the most related state-of-the-art: 1) atomic cross-chain transaction execution, and 2) blockchain data query.

### A. Atomic Cross-chain Transaction Execution

Much effort has been made to ensure atomicity and consistency of cross-chain transaction execution between two blockchains. Some recent researches use *hash time-locks contracts* (HTLCs) to execute atomic cross-chain transactions [5], [9], [10]. To be more specific, the receiver of a transaction needs to commit to the payment prior to a deadline set via smart contracts by providing the payer a cryptographic proof. Otherwise, the payment will return to the payer. HTLCs are mainly adopted in cross-chain swaps [9] and payment channels [11]. Another way to ensure the transaction atomicity between two parties is the side chain, which is detailed described in Section II. The side chain method harnesses a two-way peg mechanism to atomically conduct cross-chain asset transfer between the main chain and the side chain [12], [13]. Unfortunately, all the above approaches are only applicable to handle cross-chain transactions that happened between two parties. Due to the lack of a complete view of the cross-chain provenance dependency among multiple parties, each adopts different blockchain systems. They can hardly guarantee the correct execution order of cross-chain transactions in the multiple-chain scenarios. This will substantially affect the authenticity of data provenance query results. Focusing on the cross-chain interaction between two parties obviously cannot

meet the demands of practical application scenarios. More recently, there is an increasing focus on ensuring the atomic execution of multi-party cross-chain transactions. Herlihy et al. adopt an intermediary chain shared by all nodes to record cross-chain transaction information, prompting all parties involved in the cross-chain deal to reach a consensus on the deal execution results [10]. Zakhary et al. use a directed graph to model each cross-chain transaction and design a witness blockchain to coordinate the cross-chain transaction execution [14]. The smart contract in the witness blockchain can control and change the state of smart contracts deployed in all the participated blockchains. HyperService [5] considers the cross-chain transaction relationship. However, their solution ignores query requirements. It abstracts all transactions on the middle-layer blockchain. This solution is costly in the middle-chain while could not provide effective parallel queries. These solutions can guarantee the atomic execution of cross-chain transactions among multiple parties. Still, they essentially use the form of a relay chain to record or manage the execution of cross-chain transactions. However, the relay chain mechanism lacks scalability when the cross-chain interactions grow a lot (i.e., performance bottleneck). More importantly, the relay chain cannot authenticate cross-chain queries efficiently due to the lack of trusted cross-chain provenance dependencies.

### B. Blockchain-based Data Provenance

We can divide the current blockchain query into on-chain and off-chain, both of them focus on providing more availability in blockchain queries. Because current blockchains use LevelDB, a query-unfriendly database to store data, off-chain query strategies rearrange data for effective queries so that users can perform read and write operations as common databases. Judged by how to store data, off-chain query methods can divide into two types. The first is designing a new database to replace LevelDB [15]–[17]. They challenge the LevelDB performance and develop new database engines to solve the problems. Another solution is seeing blockchain as a store layer then adds a middle engine above the blockchain [18]–[20]. They act as translators between blockchain data and users' queries. On the other hand, on-chain query strategies focus on provide remote query more functions with trusted proofs. The first type of on-chain query optimization is data-orient [21]–[26]. They elaborately rearrange data, calculate the hash of the data under the arranged structure and then store the hash root on each blockhead, waiting for remote query nodes which do not have the original data. The second on-chain query strategy is operation-orient [27], [28]. They use smart contracts to maintain the designed structure as transactions' state. When a new transaction comes, it will trigger the smart contract to change the state. The query will perform on the state with specific query functions. Both on-chain and off-chain queries focus on a single blockchain, so they do not consider the cross-chain provenance dependency. In summary, the literature on on-chain provenance query techniques tailor for multiple blockchain scenarios still remains a vacuum. It motivates us to design Vassago that aims at providing an efficient and authenticated cross-chain data provenance mechanism.

## VIII. Discussion

*Security*: Since the nodes are divided into small groups, it is more vulnerable for malicious nodes to manipulate the blockchain. In particular, the cross-chain smart contract may be modified, resulting in an unguaranteed authenticity of the provenance query. Though our current design does not address this problem, it can be solved by adding a *Trusted Execution Environment* (TEE) [29]. Except for this, any other eligible modification of the smart contract can only be performed when all nodes participating in DB reach an agreement.

*Scalability*: Single-chain data provenance mechanism falls short in a multi-party supply chain scenario, due to significant storage overhead and inefficient provenance query performance. By dividing the nodes into different groups, Vassago performs much better in scalability and can be easily applied in large-scale scenarios.

*Usability*: In this paper, we focus on the data provenance for cross-chain transactions. However, Vassago is a general framework and can be used in many other use cases where cross-chain data queries are required.

## IX. Conclusion

Blockchain technology has shown its promises for traceable and transparent data provenance in an untrusted environment. Today's single-chain provenance query has been developed with increasing prevalence. However, it can neither support data authenticity of cross-chain transactions across multiple blockchains, nor efficient performance on different underlying blockchains. To that end, we introduce Vassago, which exploits the dependencies of cross-chain transactions to solve those problems in a multi-chain scenario. To our knowledge, Vassago is the first efficient and authenticated cross-chain provenance query mechanism. Our experimental results demonstrate Vassago can provide high efficiency without harming either throughput and storage in the worst case.

## Acknowledgment

This work was supported by Key-Area Research and Development Program of Guangdong Province under Grant No. 2020B0101090005 and National Nature Science Foundation of China under Grant No. 62072197. Jiang Xiao is the corresponding author of the paper.

## REFERENCES


[1] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, W. Cocco, Sharon, and J. Yellick, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of EuroSys conference*, 2018, pp. 1–15.

[2] Btcrelay. [Online]. Available: http://btcrelay.org/

[3] lisk. [Online]. Available: https://lisk.io//

[4] elements. [Online]. Available: https://elementsproject.org/

[5] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, and Y.-C. Hu, "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proceedings of ACM Conference on Computer and Communications Security*. ACM, 2019, pp. 549–566.

[6] M. J. Amiri, D. Agrawal, and A. E. A. Shrira, "Caper: A cross-application permissioned blockchain," in *Proceedings of the VLDB Endowment*, vol. 12, no. 11, 2019, pp. 549–566.

[7] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of ACM Conference on Computer and Communications Security*. ACM, 2018, pp. 931–948.

[8] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[9] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of ACM Symposium on Principles of Distributed Computing*. ACM, 2018, pp. 245–254.

[10] M. Herlihy, B. Liskov, and L. Shrira, "Cross-chain deals and adversarial commerce," in *Proceedings of the VLDB Endowment*, vol. 13, no. 2, 2019, pp. 100–113.

[11] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016. [Online]. Available: https://lightning.network/lightning-network-paper.pdf

[12] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K. R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *Journal of Network and Computer Applications*, vol. 149, p. 102471, 2020.

[13] Rootstock. [Online]. Available: https://www.rsk.co/

[14] V. Zakhary, D. Agrawal, and A. El Abbadi, "Atomic commitment across blockchains," in *Proceedings of the VLDB Endowment*, vol. 13, no. 9, 2020, p. 1319–1331.

[15] S. Wang, T. T. A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, W. Fu, B. C. Ooi, and P. Ruan, "Forkbase: An efficient storage engine for blockchain and forkable applications," in *Proceedings of the VLDB Endowment*, vol. 11, no. 10, 2018, pp. 1137–1150.

[16] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, and P. Jayachandran, "Blockchain meets database: Design and implementation of a blockchain relational database," in *Proceedings of the VLDB Endowment*, vol. 12, no. 11, 2019, pp. 1539–1552.

[17] Y. Zhu, Z. Zhang, C. Jin, A. Zhou, and Y. Yan, "Sebdb: Semantics empowered blockchain database," in *Proceeding of IEEE International Conference on Data Engineering*. IEEE, 2019, pp. 1820–1831.

[18] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy, "Blockchaindb: A shared database on blockchains," in *Proceedings of the VLDB Endowment*, vol. 12, no. 11, 2019, pp. 1597–1609.

[19] Y. Li, K. Zheng, Y. Yan, Q. Liu, and X. Zhou, "Etherql: a query layer for blockchain system," in *Proceedings of the International Conference on Database Systems for Advanced Applications*. Springer, 2017, pp. 556–567.

[20] F. M. Schuhknecht, A. Sharma, J. Dittrich, and D. Agrawal, "Chainifydb: How to get rid of your blockchain and use your dbms instead," in *Proceeding of Annual Conference on Innovative Data Systems Research*, 2021.

[21] T. Abdessalem and G. Jomier, "Vql: Providing query efficiency and data authenticity in blockchain systems," in *Proceedings of International Conference on Data Mining Workshops*. IEEE, 2019, pp. 1–6.

[22] X. Dai, J. Xiao, W. Yang, C. Wang, J. Chang, R. Han, and H. Jin, "Lvq: A lightweight verifiable query approach for transaction history in bitcoin," in *Proceedings of IEEE International Conference on Distributed Computing Systems*. IEEE, 2020, pp. 1020–1030.

[23] C. Xu, C. Zhang, and J. Xu, "Vchain: Enabling verifiable boolean range queries over blockchain databases," in *Proceedings of ACM Special Interest Group on Management of Data*. ACM, 2019, pp. 141–158.

[24] D. Liu, J. Ni, C. Huang, X. Lin, and X. S. Shen, "Secure and efficient distributed network provenance for iot: A blockchain-based approach," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7564–7574, 2020.

[25] M. Sigwart, M. Borkowski, M. Peise, S. Schulte, and S. Tai, "A secure and extensible blockchain-based data provenance framework for the internet of things," *Personal and Ubiquitous Computing*, vol. 6, no. 8, pp. 1–15, 2020.

[26] P. Cui, J. Dixon, U. Guin, and D. Dimase, "A blockchain-based framework for supply chain provenance," *IEEE Access*, vol. 7, pp. 157 113–157 125, 2019.

[27] C. Zhang, C. Xu, J. Xu, Y. Tang, and B. Choi, "Gem*2-tree: A gas-efficient structure for authenticated range queries in blockchain," in *Proceedings of IEEE International Conference on Data Engineering*. IEEE, 2019, pp. 842–853.

[28] P. Ruan, G. Chen, T. T. A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang, "Fine-grained, secure and efficient data provenance on blockchain systems," in *Proceedings of the VLDB Endowment*, vol. 12, no. 9, 2019, p. 975–988.

[29] S. Pang, Q. Shao, Z. Zhang, and C. Jin, "Authqx: Enabling authenticated query over blockchain via intel sgx," in *Proceedings of International Conference on Database Systems for Advanced Applications*. Springer, 2020, pp. 727–731.