

vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases

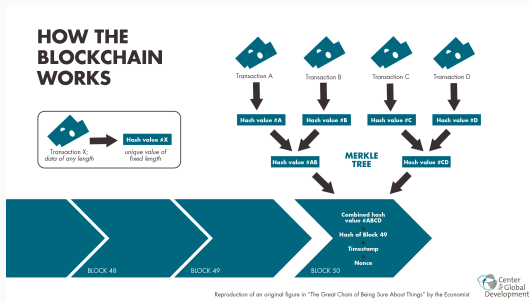
Cheng Xu Ce Zhang Jianliang Xu
{chengxu, cezhang, xujl}@comp.hkbu.edu.hk

July 2, 2019 @ SIGMOD '19

Department of Computer Science
Hong Kong Baptist University

Background

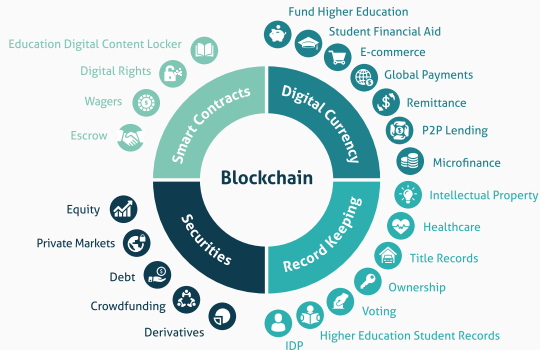
- Blockchain: Append-only data structure collectively maintained by a network of (untrusted) nodes
 - Hash chain
 - Consensus
 - Immutability
 - Decentralization



Blockchain Structure [Credit: Wikipedia]

Background

- **Blockchain: Append-only data structure** collectively maintained by a network of (untrusted) nodes
 - Hash chain
 - Consensus
 - Immutability
 - Decentralization
- A wide range of applications
 - Digital identities
 - Decentralized notary
 - Distributed storage
 - Smart Contracts
 - ...



Blockchain Applications [Credit: FAHM Technology Partners]

Blockchain Database Solutions

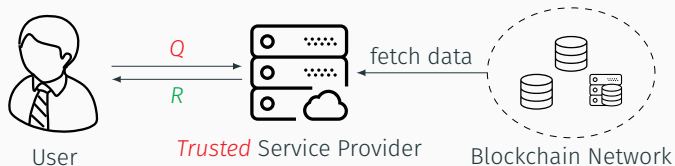
- Increasing demand to search the data stored in blockchains
- Blockchain database solutions to support SQL-like queries



Blockchain Database Solutions

Blockchain Database Solutions

- Increasing demand to search the data stored in blockchains
- Blockchain database solutions to support SQL-like queries



Workflow of Existing Solutions

- **Issue:** relying on a trusted party who can faithfully answer user queries

Secure Blockchain Search

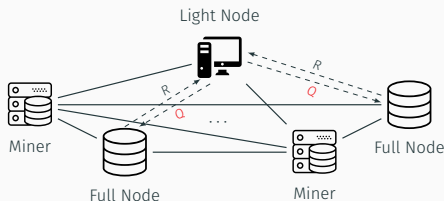
- The assumption of trusted party may not always hold

Secure Blockchain Search

- The assumption of trusted party may not always hold
- **Basic solution** to integrity-assured blockchain search
 - Becoming **full node**
 - High cost
 - **Storage**: to store a complete replicate (240 GB for Bitcoin as of June 2019)
 - **Computation**: to verify the consensus proofs
 - **Network**: to synchronize with the network

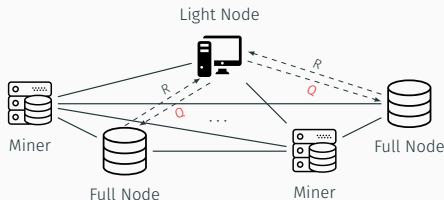
Secure Blockchain Search

- The assumption of trusted party may not always hold
- **Basic solution** to integrity-assured blockchain search
 - Becoming **full node**
 - High cost
 - **Storage**: to store a complete replicate (240 GB for Bitcoin as of June 2019)
 - **Computation**: to verify the consensus proofs
 - **Network**: to synchronize with the network
- **Better Solution**: becoming **light node** and outsource computation
 - Low cost: maintaining block headers only (< 50 MB for Bitcoin)



Secure Blockchain Search

- The assumption of trusted party may not always hold
- **Basic solution** to integrity-assured blockchain search
 - Becoming **full node**
 - High cost
 - **Storage**: to store a complete replicate (240 GB for Bitcoin as of June 2019)
 - **Computation**: to verify the consensus proofs
 - **Network**: to synchronize with the network
- **Better Solution**: becoming **light node** and outsource computation
 - Low cost: maintaining block headers only (< 50 MB for Bitcoin)



- **Challenge**: how to maintain query integrity?

Solution #1: Smart Contract

- A **trusted program** to execute **user-defined computation** upon the blockchain
 - Smart Contract reads and writes blockchain data
 - Execution integrity is ensured by the consensus protocol
- Offer trusted storage and computation capabilities
- Function as a **trusted virtual machine**

	Traditional Computer	Blockchain VM
Storage	RAM	Blockchain
Computation	CPU	Smart Contract

Solution #1: Smart Contract

- Leverage **Smart Contract** for trusted computation
 - Users submit query parameters to blockchain
 - Miners execute computation and write results into blockchain
 - Users read results from blockchain



[Credit: Oscar W]

S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *IEEE INFOCOM*, Honolulu, HI, USA, 2018, pp. 792–800.

Solution #1: Smart Contract

- Leverage **Smart Contract** for trusted computation
 - Users submit query parameters to blockchain
 - Miners execute computation and write results into blockchain
 - Users read results from blockchain

- **Drawbacks**

- **Long latency**: long time for consensus protocol to confirm a block
- **Poor scalability**: transaction rate of the blockchain is limited
- **Privacy concern**: query history is permanently and publicly stored in blockchain
- **High cost**: executing smart contract in ETH requires paying gas to miners
(*INFOCOM 2018 requires 4 201 232 gas = 0.18 Ether = 24 USD per query*)



[Credit: Oscar W]

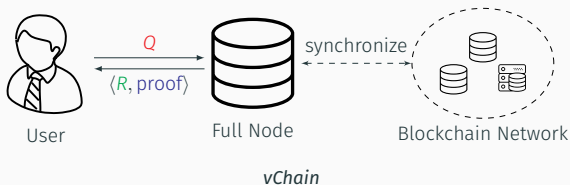
S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, "Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization," in *IEEE INFOCOM*, Honolulu, HI, USA, 2018, pp. 792–800.

Solution #2: Verifiable Computation

- Verifiable Computation (VC)
 - Computation is outsourced to untrusted service provider
 - Service provider returns results with cryptographic proof
 - Users verify integrity of results using the proof

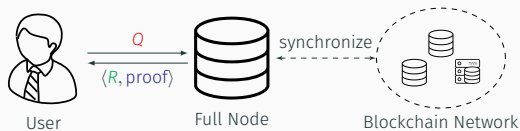
Solution #2: Verifiable Computation

- **Verifiable Computation (VC)**
 - Computation is outsourced to untrusted service provider
 - Service provider returns results with cryptographic proof
 - Users verify integrity of results using the proof
- **Outsource** queries to full node and **verify** the results using VC
 - General VC: **Expressive** but high overhead
 - Authenticated Data Structure (ADS)-based VC: **Efficient** but requiring customized designs



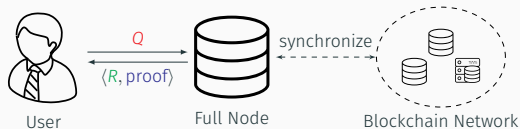
Our Solution: vChain

- **Problem:** Integrity-assured Search over Blockchain Data



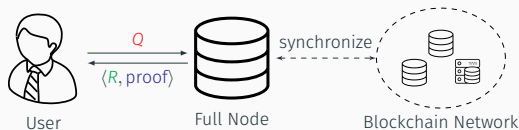
Our Solution: vChain

- **Problem:** Integrity-assured Search over Blockchain Data
- **System Model:**
 - Users become **light nodes**
 - Queries are outsourced to **full nodes**



Our Solution: vChain

- **Problem:** Integrity-assured Search over Blockchain Data
- **System Model:**
 - Users become **light nodes**
 - Queries are outsourced to **full nodes**
- **Full node not trusted**
 - Program glitches
 - Security vulnerabilities
 - Commercial interest
 - ...



Our Solution: vChain

- **Problem:** Integrity-assured Search over Blockchain Data

- **System Model:**

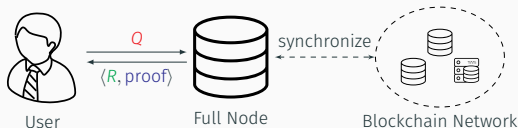
- Users become **light nodes**
- Queries are outsourced to **full nodes**

- **Full node not trusted**

- Program glitches
- Security vulnerabilities
- Commercial interest
- ...

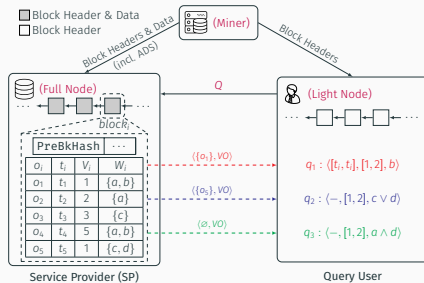
- **Security requirements**

- **Soundness:** none of the objects returned as results have been tampered with and all of them satisfy the query conditions
- **Completeness:** no valid result is missing regarding the query conditions



vChain — System Overview

- **Miner:** constructs each block with additional **ADS** to achieve VC scheme
- **Service Provider:** is a full node and computes the **results** with the verification object (**VO**)
- **Query User:** is a light node; uses the **VO** and **block header** to verify the results



- Data Model

- Each block contains several temporal objects $\{o_1, o_2, \dots, o_n\}$
- o_i is represented by $\langle t_i, V_i, W_i \rangle$
(*timestamp, multi-dimensional vector, set valued attribute*)

- Boolean Range Queries

- Find all Bitcoin transactions happening in certain period
Tx: $\langle \text{time, transfer amount, \{"send address", "receive address"\}} \rangle$
 $q = \langle [2018-05, 2018-06], [10, +\infty], \text{"send:1FFYc"} \wedge \text{"receive:2DAAf"} \rangle$
- Subscribe to car rental messages with certain price and keywords
Tx: $\langle \text{time, rental price, \{"type", "model"\}} \rangle$
 $q = \langle -, [200, 250], \text{"Sedan"} \wedge (\text{"Benz"} \vee \text{"BMW"}) \rangle$

Challenges

- How to construct ADS for **unbounded** and **append-only** blockchain data?
- How to design a **one-size-fits-all** ADS scheme that supports **dynamic queries** over **arbitrary attributes**?
- How to leverage **intra/inter-block optimization** techniques to improve query efficiency?
- How to make the system **highly scalable** to a large large number of subscription queries?

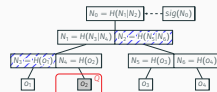
Cryptographic Building Block

- Merkle Hash Tree [Mer89]

- Support efficient membership/range queries

- **Limitations**

- An MHT supports only the query keys on which the Merkle tree is built
- MHTs do not work with set-valued attributes
- MHTs of different blocks cannot be aggregated



Merkle Hash Tree

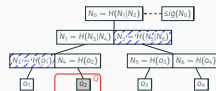
Cryptographic Building Block

- Merkle Hash Tree [Mer89]

- Support efficient membership/range queries

- **Limitations**

- An MHT supports only the query keys on which the Merkle tree is built
- MHTs do not work with set-valued attributes
- MHTs of different blocks cannot be aggregated



Merkle Hash Tree

- Cryptographic Multiset Accumulator [PTT11]

- Map a multiset to an element in cyclic multiplicative group in a collision resistant fashion
- **Utility**: prove set disjoint
- Protocols:
 - $\text{KeyGen}(1^\lambda) \rightarrow (sk, pk)$: generate keys
 - $\text{Setup}(X, pk) \rightarrow \text{acc}(X)$: return the accumulative value w.r.t. X
 - $\text{ProveDisjoint}(X_1, X_2, pk) \rightarrow \pi$:
on input two multisets X_1 and X_2 , where $X_1 \cap X_2 = \emptyset$, output a proof π
 - $\text{VerifyDisjoint}(\text{acc}(X_1), \text{acc}(X_2), \pi, pk) \rightarrow \{0, 1\}$:
on input the accumulative values $\text{acc}(X_1)$, $\text{acc}(X_2)$, and a proof π , output 1 iff $X_1 \cap X_2 = \emptyset$

Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, w_i \rangle$

Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$
- **ADS generation** (Miner)
 - Extend the block header with *AttDigest*
 - $AttDigest = acc(W_i) = Setup(W_i, pk)$
 - Constant size regardless of number of elements in W_i
 - Support $ProveDisjoint(\cdot)$ & $VerifyDisjoint(\cdot)$



Extended Block Structure

Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$
- **ADS generation** (Miner)
 - Extend the block header with *AttDigest*
 - $AttDigest = acc(W_i) = Setup(W_i, pk)$
 - Constant size regardless of number of elements in W_i
 - Support $ProveDisjoint(\cdot)$ & $VerifyDisjoint(\cdot)$
- **Verifiable Query**
 - **Match:**
 - **Mismatch:**



Extended Block Structure

Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$

- **ADS generation (Miner)**

- Extend the block header with *AttDigest*
- $AttDigest = acc(W_i) = Setup(W_i, pk)$
 - Constant size regardless of number of elements in W_i
 - Support $ProveDisjoint(\cdot)$ & $VerifyDisjoint(\cdot)$

- **Verifiable Query**

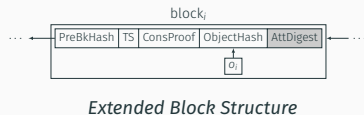
- **Match:** return o_i as a result; integrity is ensured by the *ObjectHash* in the block header
- **Mismatch:**



Extended Block Structure

Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$
- **ADS generation (Miner)**
 - Extend the block header with *AttDigest*
 - $AttDigest = acc(W_i) = Setup(W_i, pk)$
 - Constant size regardless of number of elements in W_i
 - Support $ProveDisjoint(\cdot)$ & $VerifyDisjoint(\cdot)$
- **Verifiable Query**
 - **Match:** return o_i as a result; integrity is ensured by the *ObjectHash* in the block header
 - **Mismatch:** use *AttDigest* to prove the mismatch of o_i



Basic Solution

- Consider *a single object* and *boolean query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$

- **ADS generation (Miner)**

- Extend the block header with *AttDigest*
- $AttDigest = acc(W_i) = Setup(W_i, pk)$
 - Constant size regardless of number of elements in W_i
 - Support $ProveDisjoint(\cdot)$ & $VerifyDisjoint(\cdot)$

- **Verifiable Query**

- **Match:** return o_i as a result; integrity is ensured by the *ObjectHash* in the block header
- **Mismatch:** use *AttDigest* to prove the mismatch of o_i



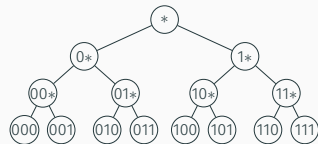
Extended Block Structure

Example of Mismatch

- Transform query condition to a list of sets: $q = \text{"Sedan"} \wedge (\text{"Benz"} \vee \text{"BMW"}) \rightarrow \{\text{"Sedan"}\}, \{\text{"Benz"}, \text{"BMW"}\}$
- Consider $o_i : \{\text{"Van"}, \text{"Benz"}\}$, we have $\{\text{"Sedan"}\} \cap \{\text{"Van"}, \text{"Benz"}\} = \emptyset$
- Apply $ProveDisjoint(\{\text{"Van"}, \text{"Benz"}\}, \{\text{"Sedan"}\}, pk)$ to compute proof π
- User retrieves $AttDigest = acc(\{\text{"Van"}, \text{"Benz"}\})$ from the block header and uses $VerifyDisjoint(AttDigest, acc(\{\text{"Sedan"}\}), \pi, pk)$ to verify the mismatch

Extension to Range Queries

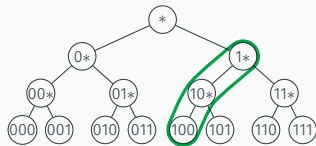
- **Idea:** transform numerical attributes into set-valued attributes



Example of Transformation

Extension to Range Queries

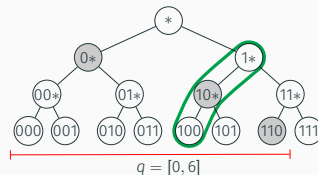
- **Idea:** transform numerical attributes into set-valued attributes
- Numerical value can be transformed into a set of binary **prefix elements**
 - **Example:** $\text{trans}(4) = \{1*, 10*, 100\}$
 - * denotes wildcard matching operator



Example of Transformation

Extension to Range Queries

- **Idea**: transform numerical attributes into set-valued attributes
- Numerical value can be transformed into a set of binary **prefix elements**
 - **Example**: $\text{trans}(4) = \{1*, 10*, 100\}$
* denotes wildcard matching operator
- Range can be transformed into an equivalent **boolean expression** using a binary tree
 - **Example**: $[0, 6] \rightarrow 0* \vee 10* \vee 110$
Equivalence set: $\{0*, 10*, 110\}$



Example of Transformation

Extension to Range Queries

- **Idea**: transform numerical attributes into set-valued attributes

- Numerical value can be transformed into a set of binary **prefix elements**

- **Example**: $\text{trans}(4) = \{1*, 10*, 100\}$
* denotes wildcard matching operator

- Range can be transformed into an equivalent **boolean expression** using a binary tree

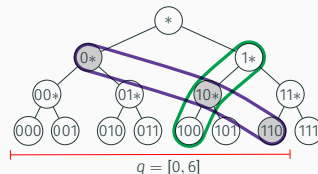
- **Example**: $[0, 6] \rightarrow 0* \vee 10* \vee 110$
Equivalence set: $\{0*, 10*, 110\}$

- **Range queries** can be processed in a similar manner as **boolean queries**

- Transform $v_i \in [\alpha, \beta] \rightarrow \text{trans}(v_i) \cap \text{EquiSet}([\alpha, \beta]) \neq \emptyset$

- **Example**:

- $4 \in [0, 6] \rightarrow \{1*, 10*, 100\} \cap \{0*, 10*, 110\} = \{10*\} \neq \emptyset$



Example of Transformation

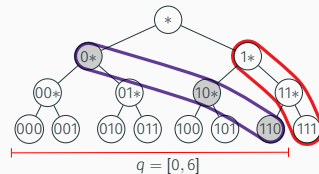
Extension to Range Queries

- **Idea:** transform numerical attributes into set-valued attributes
- Numerical value can be transformed into a set of binary **prefix elements**

- **Example:** $\text{trans}(4) = \{1*, 10*, 100\}$
* denotes wildcard matching operator

- Range can be transformed into an equivalent **boolean expression** using a binary tree

- **Example:** $[0, 6] \rightarrow 0* \vee 10* \vee 110$
Equivalence set: $\{0*, 10*, 110\}$



Example of Transformation

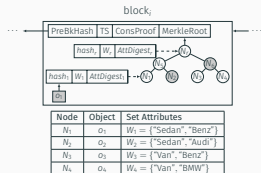
- **Range queries** can be processed in a similar manner as **boolean queries**
 - Transform $v_i \in [\alpha, \beta] \rightarrow \text{trans}(v_i) \cap \text{EquiSet}([\alpha, \beta]) \neq \emptyset$
 - **Example:**
 - $4 \in [0, 6] \rightarrow \{1*, 10*, 100\} \cap \{0*, 10*, 110\} = \{10*\} \neq \emptyset$
 - $7 \notin [0, 6] \rightarrow \{1*, 11*, 111\} \cap \{0*, 10*, 110\} = \emptyset$

Batch Verification & Subscription Queries

- **Observation**: objects may share common attributes that mismatch query condition
- **Idea**: we can aggregate them to speed up query processing

Batch Verification & Subscription Queries

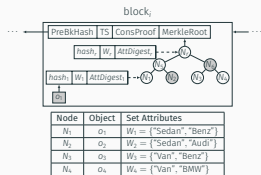
- **Observation:** objects may share common attributes that mismatch query condition
- **Idea:** we can aggregate them to speed up query processing
 - **Intra-Block Index:** aggregate objects inside same block using MHT



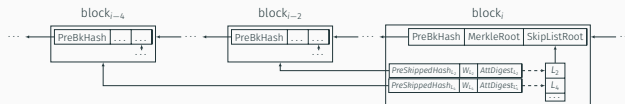
Intra-Block Index

Batch Verification & Subscription Queries

- **Observation:** objects may share common attributes that mismatch query condition
- **Idea:** we can aggregate them to speed up query processing
 - **Intra-Block Index:** aggregate objects inside same block using MHT
 - **Inter-Block Index:** aggregate objects across blocks using skip list



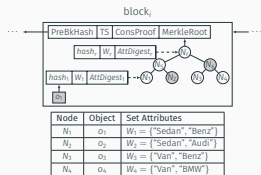
Intra-Block Index



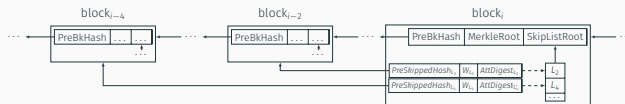
Inter-Block Index

Batch Verification & Subscription Queries

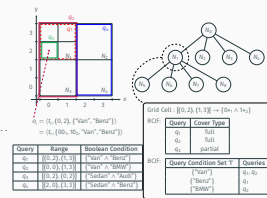
- **Observation:** objects may share common attributes that mismatch query condition
- **Idea:** we can aggregate them to speed up query processing
 - **Intra-Block Index:** aggregate objects inside same block using MHT
 - **Inter-Block Index:** aggregate objects across blocks using skip list
 - **Inverted Prefix Tree:** aggregate similar subscription queries from users



Intra-Block Index



Inter-Block Index



Inverted Prefix Tree

Performance Evaluation

- Evaluation metrics

- Query processing cost in terms of **SP CPU time**
- Query verification cost in terms of **user CPU time**
- Size of the VO** transmitted from the SP to the user

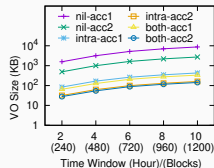
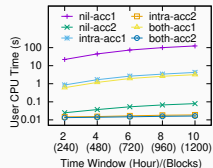
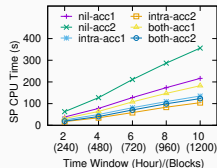
- Numerical range selectivity

- 10% for 4SQ
- 50% for ETH

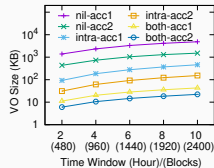
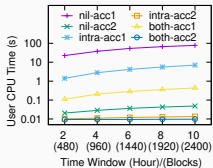
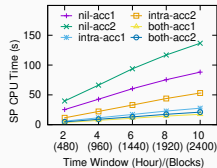
- Disjunctive Boolean function size

- 3 for 4SQ
- 9 for ETH

4SQ



ETH



Time-Window Query Performance

Thanks
Questions?

References

- [HCW+18] S. Hu, C. Cai, Q. Wang, C. Wang, X. Luo, and K. Ren, “Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization,” in *IEEE INFOCOM*, Honolulu, HI, USA, 2018, pp. 792–800.
- [ICDE19] C. Zhang, C. Xu, J. Xu, Y. Tang, and B. Choi, “GEM²-Tree: A gas-efficient structure for authenticated range queries in blockchain,” in *IEEE ICDE*, Macau SAR, China, 2019.
- [Mer89] R. C. Merkle, “A certified digital signature,” in *CRYPTO*, 1989, pp. 218–238.
- [PTT11] C. Papamanthou, R. Tamassia, and N. Triandopoulos, “Optimal verification of operations on dynamic sets,” in *CRYPTO*, Santa Barbara, CA, USA, 2011, pp. 91–110.
- [SIGMOD19] C. Xu, C. Zhang, and J. Xu, “vChain: Enabling verifiable boolean range queries over blockchain databases,” in *ACM SIGMOD*, Amsterdam, Netherlands, 2019.