Towards Searchable and Verifiable Blockchain

Cheng Xu   Ce Zhang

April 8, 2019

Department of Computer Science
Hong Kong Baptist University

Background

- Blockchain $\neq$ Bitcoin

Background

- Blockchain $\neq$ Bitcoin
- Blockchain is a distributed ledger maintained by a community of (untrusted) users
  - Decentralization
  - Consensus
  - Immutability
  - Provenance

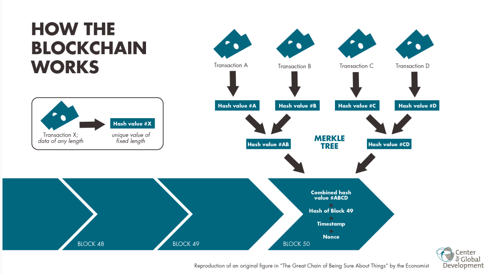


Fig. 1: Blockchain Structure [Credit: Wikipedia]

Background

- Blockchain $\neq$ Bitcoin
- Blockchain is a distributed ledger maintained by a community of (untrusted) users
    - Decentralization
    - Immutability
    - Consensus
    - Provenance
- A wide range of applications
    - Record Keeping
    - Smart Contracts
    - ...

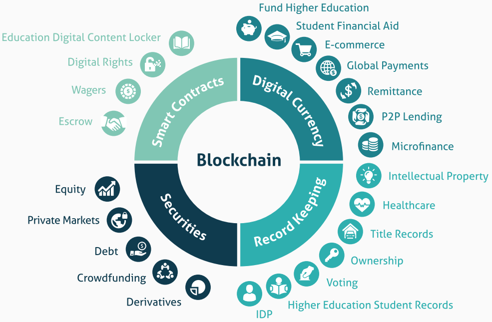


Fig. 2: Blockchain Applications [Credit: FAHM Technology Partners]

Blockchain Database Solutions

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

Fig. 3: Blockchain Database Solutions

Blockchain Database Solutions

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

Fig. 3: Blockchain Database Solutions

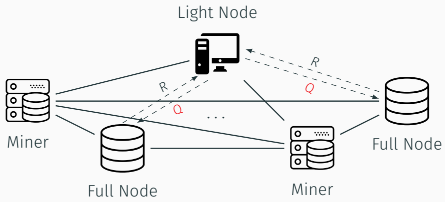
- **Issue**: relying on a trusted party who can faithfully execute user queries

Blockchain Search Problem

- Integrity assurance: query results retrieved from the blockchain should be publicly verifiable
    - Becoming full node
    - High cost
        - Storage: to store a complete replicate (240 GB for Bitcoin as of Mar 2019)
        - Computation: to verify the consensus proofs
        - Network: to synchronize with the network

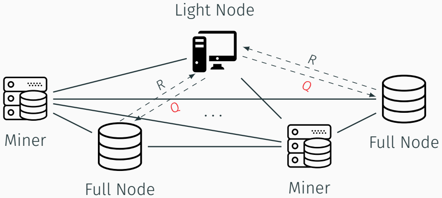
Blockchain Search Problem

- Integrity assurance: query results retrieved from the blockchain should be publicly verifiable
  - Becoming full node
  - High cost
    - Storage: to store a complete replicate (240 GB for Bitcoin as of Mar 2019)
    - Computation: to verify the consensus proofs
    - Network: to synchronize with the network
- Alternative approach: becoming light node and outsource computation
  - Low cost: maintaining block headers only (< 50 MB for Bitcoin)



Blockchain Search Problem

- Integrity assurance: query results retrieved from the blockchain should be publicly verifiable
  - Becoming full node
  - High cost
    - Storage: to store a complete replicate (240 GB for Bitcoin as of Mar 2019)
    - Computation: to verify the consensus proofs
    - Network: to synchronize with the network
- Alternative approach: becoming light node and outsource computation
  - Low cost: maintaining block headers only (< 50 MB for Bitcoin)



- Question: how to ensure integrity?

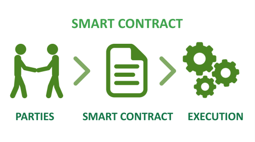
Solution #1: Smart Contract

- 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

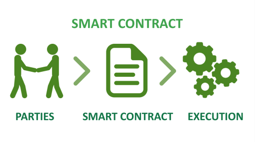


[Credit: Oscar W]

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

- Verification 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

- Verification 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 Solutions

- vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases (SIGMOD 2019)
- GEM^2-Tree: Enabling Gas-Efficient Authenticated Range Queries for Hybrid Storage in Blockchain (ICDE 2019)

vChain: Enabling Verifiable Boolean Range Queries over Blockchain Databases

Cheng Xu, Ce Zhang, and Jianliang Xu

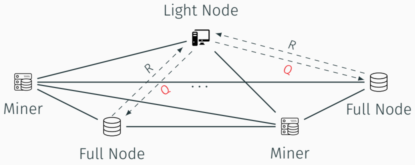
ACM SIGMOD 2019

vChain — Problem Definition

- **Problem**: integrity-assured search over blockchain data

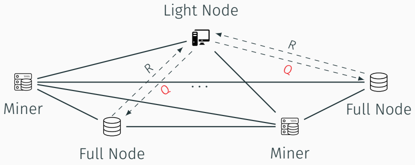
vChain — Problem Definition

- **Problem**: integrity-assured search over blockchain data
- **System Model**:
  - Users become light nodes
  - Queries are outsourced to full nodes



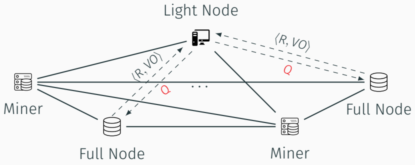
vChain — Problem Definition

- **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
  - ...



vChain — Problem Definition

- 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 window or subscription period

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

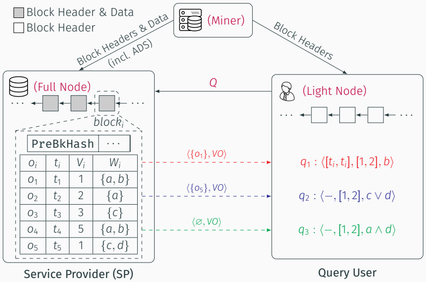


Fig. 4: System Model of vChain

vChain — Data Model & Queries

- Data Model
  - Each block contains several temporal objects $\{o_1, o_2, \ldots, o_n\}$
  - o_i is represented by $\langle t_i, V_i, W_i \rangle$
    (*timestamp, multi-dimensional vector, set valued attribute*)
- Boolean Range Queries
  - Time-window queries:
    $q = \langle [2018\text{-}05, 2018\text{-}06], [10, +\infty], \text{“send:1FFYc”} \wedge \text{“receive:2DAAf”} \rangle$
  - Subscription queries:
    $q = \langle -, [200, 250], \text{“Sedan”} \wedge (\text{“Benz”} \vee \text{“BMW”}) \rangle$

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

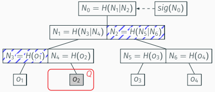


Fig. 5: Merkle Hash Tree

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

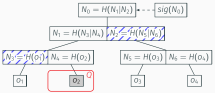


Fig. 5: Merkle Hash Tree

- **Cryptographic Multiset Accumulator** [PTT11]
  - Map a multiset to an element in cyclic multiplicative group in a collision resistant fashion
  - Utility: prove set disjoint
  - Protcols:
    - $\text{KeyGen}(1^\lambda) \rightarrow (sk, pk)$: 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 $X_1 \cap X_2 = \varnothing$, 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 $X_1 \cap X_2 = \varnothing$

Basic Solution

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

Basic Solution

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

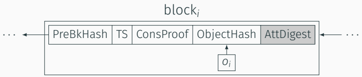


Fig. 6: Extended Block Structure

Basic Solution

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

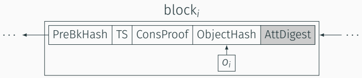


Fig. 6: Extended Block Structure

Basic Solution

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

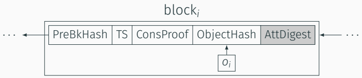


Fig. 6: Extended Block Structure

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

Basic Solution

- Consider *a single object* and *boolean time-window query*
- Each block stores a single object $o_i = \langle t_i, W_i \rangle$
- **ADS generation** (Miner)
  - Extend the block header with *AttDigest*
  - $AttDigest = acc(W_i) = \text{Setup}(W_i, pk)$
    - 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

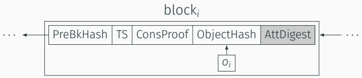


Fig. 6: Extended Block Structure

Basic Solution

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

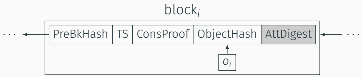


Fig. 6: Extended Block Structure

- **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

Example of Mismatch

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

Basic Solution

- Support time-window queries
  - Find the blocks whose timestamp is within the query window
  - Invoke previous algorithm for each object in theses blocks

Example

- $q =$ "Sedan" $\wedge$ ("Benz" $\vee$ "BMW")
- Objects within the time window o_1 : {"Van", "Benz"}, o_2 : {"Sedan", "Audi"}, o_3 : {"Van", "Benz"}
- Query processing
  - o_1 is returned as a result
  - ProveDisjoint($\cdot$) is applied for o_2 and o_3
    - Mismatch condition "Benz" $\vee$ "BMW" for o_2
    - Mismatch condition "Sedan" for o_3

Extension to Range Queries

- Idea: transform numerical attributes into set-valued attributes

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

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 a equivalent boolean expression using a binary tree
  - Example: $[0, 6] \rightarrow 0* \ \vee \ 10* \ \vee \ 110$
    Equivalence set: $\{0*, 10*, 110\}$

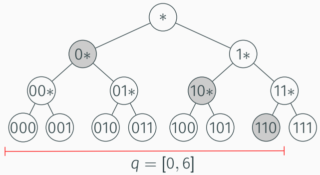


Fig. 7: Example of Transformation

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 a equivalent boolean expression using a binary tree
  - Example: $[0, 6] \rightarrow 0* \;\vee\; 10* \;\vee\; 110$

    Equivalence set: $\{0*, 10*, 110\}$

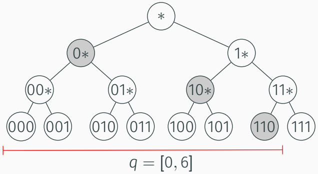


Fig. 7: Example of Transformation

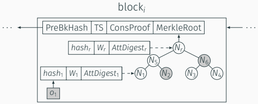
- Range queries can be processed in a similar manner as Boolean queries
  - Transform $v_i \in [\alpha, \beta] \rightarrow \text{trans}(v_i) \cap \text{EquiSet}([\alpha, \beta]) \neq \varnothing$
  - Example:
    - $4 \in [0, 6] \rightarrow \{1*, 10*, 100\} \cap \{0*, 10*, 110\} = \{10*\} \neq \varnothing$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

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



Node	Object	Set Attributes
N_1	o_1	$W_1 = \{$"Sedan", "Benz"$\}$
N_2	o_2	$W_2 = \{$"Sedan", "Audi"$\}$
N_3	o_3	$W_3 = \{$"Van", "Benz"$\}$
N_4	o_4	$W_4 = \{$"Van", "BMW"$\}$

Fig. 8: 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

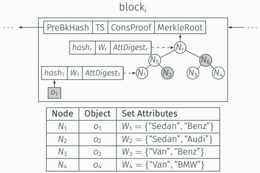


Fig. 8: Intra-Block Index

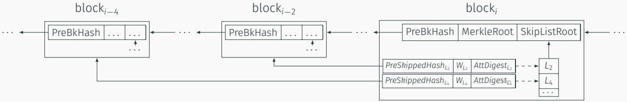
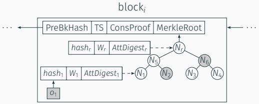


Fig. 9: Inter-Block Index

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



Node	Object	Set Attributes
N_1	o_1	W_1 = {"Sedan", "Benz"}
N_2	o_2	W_2 = {"Sedan", "Audi"}
N_3	o_3	W_3 = {"Van", "Benz"}
N_4	o_4	W_4 = {"Van", "BMW"}

Fig. 8: Intra-Block Index

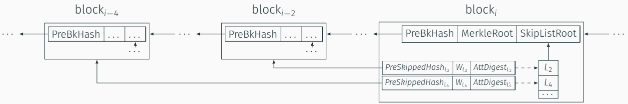


Fig. 9: Inter-Block Index

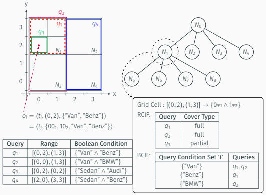


Fig. 10: Inverted Prefix Tree

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 & WX
  - 50% for ETH
- Disjunctive Boolean function size
  - 3 for 4SQ & WX
  - 9 for ETH

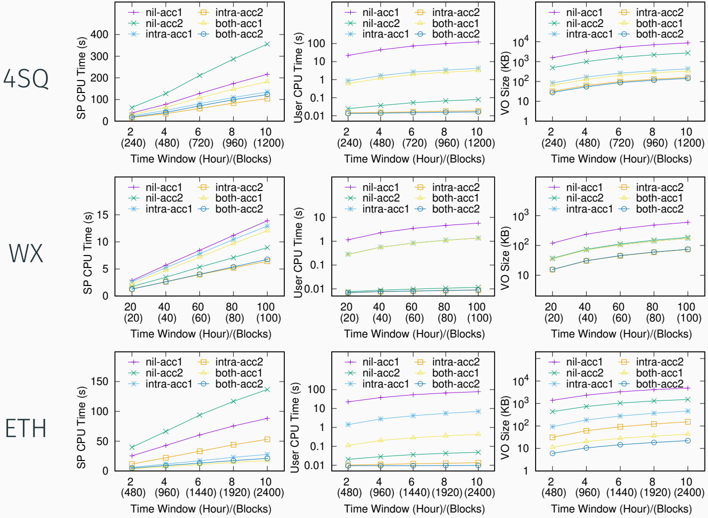


Fig. 11: Time-Window Query Performance

GEM^2-Tree: Enabling Gas-Efficient Authenticated Range Queries for Hybrid Storage in Blockchain

Ce Zhang, Cheng Xu, Jianliang Xu, Yuzhe Tang, and Byron Choi

IEEE ICDE 2019

GEM^2-Tree

- More details
  - Section: Research (14) — Query Processing, Indexing and Optimization
  - Time: 14:35–16:05, April 10, Wednesday
  - Location: 7004

GEM^2-Tree

- Storing data on chain is not scalable

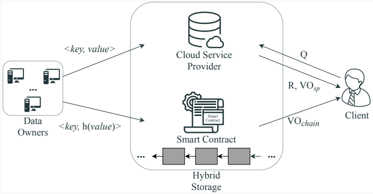


Fig. 12: Authenticated Query Framework in Hybrid-Storage Blockchain

- **More details**
  - Section: Research (14) — Query Processing, Indexing and Optimization
  - Time: 14:35–16:05, April 10, Wednesday
  - Location: 7004

GEM^2-Tree

- Storing data on chain is not scalable
- Hybrid storage:
  - Raw data is stored off-chain
  - A hash of the data is keep on chain to ensure integrity
  - Smart contract maintains on-chain index to facilitate authenticated query processing

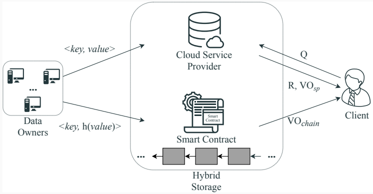


Fig. 12: Authenticated Query Framework in Hybrid-Storage Blockchain

- More details
  - Section: Research (14) — Query Processing, Indexing and Optimization
  - Time: 14:35–16:05, April 10, Wednesday
  - Location: 7004

- Storing data on chain is not scalable
- Hybrid storage:
  - Raw data is stored off-chain
  - A hash of the data is keep on chain to ensure integrity
  - Smart contract maintains on-chain index to facilitate authenticated query processing
- Question: How to reduce transaction fee a.k.a *gas*?
- More details
  - Section: Research (14) — Query Processing, Indexing and Optimization
  - Time: 14:35–16:05, April 10, Wednesday
  - Location: 7004

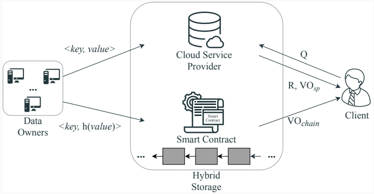


Fig. 12: Authenticated Query Framework in Hybrid-Storage Blockchain

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.

[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.

[XZX19] C. Xu, C. Zhang, and J. Xu, "vChain: Enabling verifiable boolean range queries over blockchain databases," in *ACM SIGMOD*, Amsterdam, Netherlands, 2019.

[ZXX+19] 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 *IEEE ICDE*, Macau SAR, China, 2019.