MtDB: A Decentralized Multi-Tenant Database for Secure Data Sharing

Showkot Hossain*, Wenyi Tang*, Changhao Chenli†, Haijian Sun‡,
WenZhan Song‡, Seokki Lee§, Mic Bowman¶, and Taeho Jung*
*University of Notre Dame, USA
†Indiana Institute of Technology, USA
‡University of Georgia, USA
§University of Cincinnati, USA
¶Intel Corporation, USA

***Abstract*—Healthcare data sharing is fundamental for advancing medical research and enhancing patient care, yet it faces significant challenges in privacy, data ownership, and interoperability due to fragmented data silos across institutions and strict regulations (e.g., GDPR, HIPAA). Patients possess distributed records across multiple hospitals, each maintaining autonomous databases. Access to consolidated records by secondary entities mandates explicit patient consent while ensuring strict isolation between multi-tenant datasets, requiring fine-grained access control across organizational boundaries. Existing solutions exhibit critical limitations: blockchain-based databases lack robust fine-grained cryptographic enforcement of dynamic access policies, while TEE-enhanced systems suffer from synchronization overhead and poor scalability in distributed deployments. To bridge these gaps, we propose MtDB, a novel decentralized database architecture addressing secure data sharing in multi-tenant database ecosystems. MtDB employs blockchain for metadata coordination and sharing, IPFS for distributed data addressing, a universal SQL query interface for data access, and Intel SGX for integrity-protected query execution with enforced access control. We provide an open-source implementation demonstrating MtDB's capabilities for secure, patient-centric healthcare data sharing while preserving ownership and enforcing policies. Experimental results show MtDB achieves 35 milliseconds query latency for indexed queries over 400M multi-tenant medical records while maintaining cryptographic security guarantees, with only 1.2–1.3× performance overhead compared to non-secure baselines.**

***Index Terms*—Decentralized Data Sharing, Access Control, Trusted Execution Environment, Blockchain.**

I. Introduction

Healthcare data sharing is pivotal for advancing medical research and improving patient outcomes, yet it introduces significant challenges regarding privacy, ownership, and interoperability. With various legislations on digital privacy (e.g., GDPR [1], HIPAA [2]) being released, there is growing need to respect data ownership when using collected personal data. In the scenario of medical and healthcare data, this means patients hold ownership of their data, and any inquiry to use the data should have explicit consent from its owner. However, medical data is often generated through interactions between individuals and medical institutions and is maintained by these institutions in their own silos with data collected from different users. Thus, ensuring individual consent and enforcement of owner-defined access control policies during personal medical data sharing from multi-tenant database scenarios faces additional challenges.

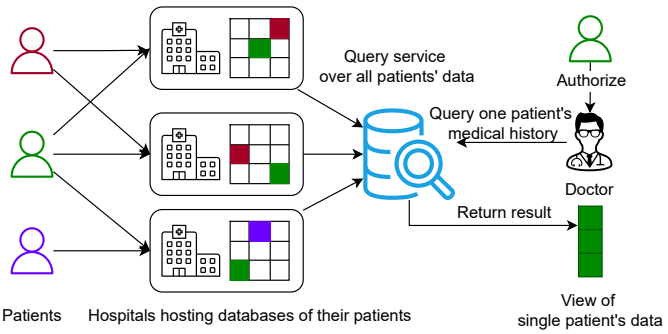


Fig. 1: Example of a multi-tenant database for healthcare data sharing. The query service should return the patient's data from all data partitions maintained in different hospitals.

We consider a representative scenario for medical data sharing, where a patient may visit different hospitals specializing in treating different conditions (as shown in Figure 1). For a specific patient, every visit to a hospital generates a medical record managed by that hospital. To prescribe appropriate treatment for a new condition, a physician may need to review the patient's complete medical history, including past diagnoses and previous prescriptions. However, requesting the patient's complete history is not trivial when records are distributed across various data silos or partitions.

This scenario requires three key capabilities: First, since patient records are distributed in partitions managed by different hospitals, querying the whole history requires proper coordination. This implies a universal query interface that processes requests over different partitions of the logical database. Second, even single hospital databases have multi-tenant structures, where every entry may have different ownership or owner-specific access policies. This requires that query processing at every data silo strictly protects data from leaking to unauthorized parties. Third, the integrity of query execution must be guaranteed during processing over every partition to ensure owner-defined access control policies are respected.

Existing solutions fail to address these multifaceted chal-

lenges of healthcare data sharing in multi-tenant scenarios, primarily due to fundamental limitations in managing distributed data ownership, enforcing row-level access control, and providing efficient SQL query interfaces. While distributed databases [3]–[5] provide scalability and fault tolerance, they inherently rely on trusted administrators and lack cryptographic enforcement of row-level access policies, which is essential for multi-tenant healthcare deployments. Similarly, TEE-based secure databases [6]–[9] deliver robust security but do not support fine-grained, patient-specific access control required in decentralized multi-owner environments. Blockchain-based systems [10]–[12] offer decentralization but lack cryptographic policy enforcement and secure in-enclave computation capabilities. Additionally, existing IPFS indexing frameworks [13]–[15] primarily support plaintext, NoSQL queries and cannot efficiently handle encrypted relational queries. Critically, no prior solution simultaneously addresses: 1) fine-grained, row-level access control in multi-tenant settings; 2) efficient SQL querying over encrypted IPFS-resident data; and 3) seamless connection among distributed data silos.

To bridge these gaps, we propose MtDB, a novel SQL-based decentralized database architecture designed specifically to enable secure, patient-centric data sharing in decentralized multi-tenant healthcare environments. MtDB connects distributed data silos to present a universal SQL-based logical database for sharing healthcare data. To make a secure data sharing service, MtDB allows users to define specific access control policies over data they own in the multi-tenant database, and these policies are enforced during query execution, no matter where the query is served, therefore respecting individual ownership in the multi-tenant scenario. These policies are directly applied to the original SQL request through a query rewriter, and the rewritten query is executed in any MtDB nodes, which use a TEE-backed query engine to provide integrity and security. To maintain the logical database for MtDB, we designed an efficient delta-based on-chain index system to coordinate partitions hosted by different hospitals and agencies with minimized on-chain cost. Healthcare agencies can update and retrieve table schema, index, IPFS addressable content identifiers, and other metadata from the public blockchain through MtDB nodes with security and integrity guarantees provided by TEE. Our contributions include:

- We define data sharing under multi-tenant database scenarios, widely applicable in real-world healthcare data sharing.
- We designed MtDB, a SQL-based decentralized secure database architecture addressing multi-tenant data sharing challenges through user-defined access control policies enforced by TEE-based query engines and efficient delta-based on-chain indexing.
- We provide an open-source implementation with comprehensive evaluations demonstrating MtDB's capability and efficiency. Our code: https://tinyurl.com/mtdb2025.

II. Related Work

Traditional distributed databases such as Apache Cassandra [4] and CockroachDB [3] offer scalability and fault tolerance but rely on centralized trust models, assuming fully trusted administrators and infrastructure. This limits their suitability for multi-tenant healthcare environments requiring strong data isolation and cryptographic access enforcement. Google Spanner [5] achieves global consistency using atomic clocks but depends entirely on Google's infrastructure. These limitations have prompted interest in decentralized models leveraging cryptographic techniques and trusted hardware to enforce access control without centralized trust.

Hardware-assisted TEEs, such as Intel SGX [16], [17], enable secure computation with less overhead than software-based cryptography. EnclaveDB [9] introduced in-enclave SQL execution but relies on pre-compiled queries and centralized trust. StealthDB [18] adds encrypted SQL support on PostgreSQL but lacks decentralization. SecuDB [6] offers tamper resistance and fine-grained masking within a single domain. QShield [8] supports multi-user access control for outsourced queries but does not address decentralized, cross-institution data sharing.

To enable analytics over encrypted data, frameworks such as CryptDB [19], Opaque [20], and Enigma [21] have been proposed–most assuming centralized infrastructures. CryptDB uses layered encryption for SQL queries but relies on a trusted proxy, exposing access patterns with limited SQL support. Opaque integrates SGX with Apache Spark to protect data flows but assumes centralized control. Enigma combines MPC with blockchain for decentralization but lacks efficient indexing and data management.

Blockchain-based databases aim to minimize trust through decentralization but struggle to balance performance and privacy. Systems like BigChainDB [22], OrbitDB [23], and Web3DB [10] incorporate blockchain for immutability but lack cryptographic access control and TEE integration for secure computation. Web3DB advances decentralized relational querying with blockchain-managed access control but does not provide enclave-based secure execution. IPFS supports decentralized storage via content-addressing, but secure and efficient querying remains challenging. Existing indexing schemes [13]–[15] target NoSQL-style queries over plaintext data and lack SQL or fine-grained access control support. Although blockchain-based access control schemes [24]–[26] help define policies, they do not support confidential query execution or fine-grained access controls required in healthcare.

III. Preliminaries

A. IPFS

InterPlanetary File System (IPFS) is a peer-to-peer distributed file system that uses content-addressing, where files are identified by cryptographic hashes called Content Identifiers (CIDs), ensuring data integrity and immutability. IPFS enables distributed hosting, reduces reliance on centralized servers, and improves redundancy through mechanisms like

pinning. The architecture integrates Distributed Hash Table (DHT), Bitswap (data exchange protocol), and Merkle Directed Acyclic Graphs (DAGs) for efficient content discovery and transfer. IPFS utilizes libp2p for network communication and IPLD for linking content structures, creating a resilient, scalable, and censorship-resistant platform.

In MtDB, we treat IPFS as the persistent storage layer where database files are distributed and addressed by CID. IPFS manages storage and transmission with its internal mechanisms to ensure files can be retrieved from the network using a given CID.

B. Blockchain and Smart Contract

Blockchain is a decentralized, distributed ledger system ensuring data integrity through cryptographic principles and consensus mechanisms. Each block contains a cryptographic hash of the previous block, timestamp, and transactional data, forming an immutable tamper-resistant sequence. Validated transactions are organized as a Merkle tree and packed into blocks based on the consensus protocol, then validated by blockchain maintainers and broadcast network-wide.

Smart contracts are self-executing programs deployed on blockchains that autonomously enforce predefined rules when conditions are met. They leverage blockchain's decentralized architecture to enable trustless transactions and provide deterministic execution. Core properties include autonomy (automatic execution), tamper-proofness (code immutability post-deployment), and auditability (transparent on-chain operations).

C. Trusted Execution Environment

Trusted Execution Environments (TEEs) provide isolated execution spaces for sensitive computations, shielding code and data from compromised operating systems and privileged software through hardware-enforced memory encryption and access controls. TEEs partition system resources into "trusted" and "untrusted" domains, guaranteeing confidentiality and integrity even when the host environment is adversarial.

We use Intel SGX as a core component to build MtDB. SGX's security derives from isolated execution: enclave memory (Enclave Page Cache) is encrypted and integrity-protected using the Memory Encryption Engine, accessible only to authorized enclave code. SGX provides remote attestation, allowing remote parties to cryptographically verify an enclave's identity and integrity via hardware-signed quotes, establishing secure channels for provisioning secrets exclusively to validated enclaves. We exploit these properties to build a confidential computing framework where sensitive database operations and access control enforcement execute within attested enclaves, guaranteeing code authenticity and data secrecy against privileged attackers.

IV. System Overview and Models

In this section, we describe the scenario MtDB mainly deals with and the corresponding assumptions of MtDB. We then give an overview of the architecture and the system workflow of MtDB.

A. Distributed Multi-Tenant Database Management

We define Distributed Multi-Tenant Database Management as a decentralized data ecosystem comprising three entity classes:

- Users ($\mathcal{U}$): A large set of individual data owners (e.g., patients), each generating personal data records through interactions with service managers.
- Service Managers ($\mathcal{M}$): A smaller set of entities (e.g., hospitals, pharmacies) taking interactions with users as well as collecting, storing, and managing partitioned databases containing user-generated records.
- Queriers ($\mathcal{Q}$): Anyone who tries to query the ecosystem among data collected from all service managers.

The architecture of MtDB satisfies the following core properties:

- Every record r is owned by the user $u \in \mathcal{U}$ from whom it originated, irrespective of the managing entity $m \in \mathcal{M}$ that hosts it. Even though the database management systems usually provide attribute-level access control for different users, MtDB demands a more fine-grained access control.
- Each $m_i \in \mathcal{M}$ maintains a physically partitioned database DB_i, with the collective $\bigcup_{i=1}^{|\mathcal{M}|} DB_i$ forming a logical database. No central data aggregation occurs.
- Ownership is determined at the record level, where a single user's records may span multiple DB_i due to distinct interactions with different service managers.

To allow seamless data sharing across different data managers and provide a Unified Query Interface, the table definitions will be publicly available for data managers to format their data and for users to construct query requests. With the unified query interface, users can interact with the data sharing ecosystem constructed by multiple data managers as if interacting with a single database infrastructure. (e.g., searching a patient's data from all hospitals will be as simple as requesting a query: `SELECT * FROM table WHERE user_name = John Doe`), subject to ownership-based authorization.

B. System Components

As shown in Figure 2, MtDB employs a decentralized architecture integrating trusted execution environments (Intel SGX), blockchain, and IPFS to enable secure multi-tenant healthcare data sharing. Each MtDB node runs an Intel SGX enclave and connects to the same blockchain (Ethereum in our implementation) and IPFS network. Every MtDB node can individually serve queries from any user with the same logical database interface. Specifically, MtDB node uses the Query Interface to accept SQL queries via HTTPS and routes them to the In-Enclave Engine to process. The In-Enclave Engine will first check if there is explicit consent from the user, who owns the data that is being queried. This could be verified through a one-time consent from the user's signature, verifiable credentials [27], or pre-defined access control policy entry shared through the blockchain. Upon successful verification,

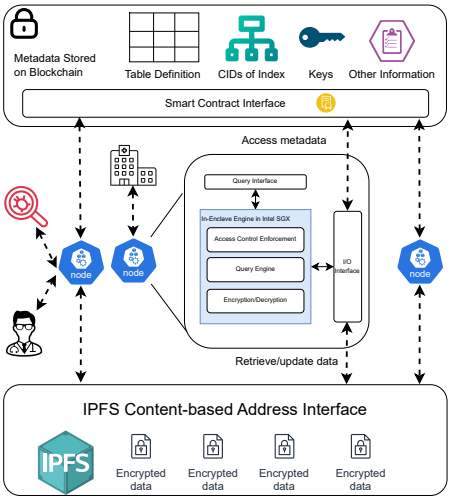


Fig. 2: Architecture overview of MtDB

the In-Enclave Engine will retrieve the metadata through the smart contract interface on the blockchain and corresponding database partitions from the IPFS. Both connections will go through the IO Interface, which handles the host OS I/O and the network connection.

In MtDB, the blockchain acts like a bulletin board for all MtDB nodes to share metadata about the multi-tenant database. The metadata includes:

- The table definitions, which can be referred to by all healthcare agencies to format their data and join the data sharing. It also helps users construct their query requests;
- The CIDs to the delta index updates, which allows the MtDB nodes to locate the index files from IPFS and construct their local copy of the index, allowing efficient query execution with minimum partition retrievals;
- The decryption keys of partitions, which allow a MtDB node to retrieve partitions and decrypt to serve the query.
- Other information, for example, the identity-related information of users can be stored as metadata, since MtDB does not handle identity management internally. One can use any mechanism by storing certificates from PKI or anonymous credentials [28] on the blockchain and perform the identity verification inside of MtDB node in SGX enclave. Another example would be the access control policies (or the reference to the policies) defined by the owner. These policies could be pre-defined and shared via blockchain, so that the In-Enclave Engine can retrieve and apply them to run the query.

Note that all of the metadata on the blockchain will be stored through the processing of the SGX enclave of MtDB nodes. This allows metadata to be encrypted within the SGX enclave before being recorded on the blockchain, ensuring that only authorized SGX enclaves–provisioned with the appropriate symmetric key via a secure channel established through remote attestation–can decrypt and access the metadata.

C. MtDB Dataflow

1) Data Uploading with Index Construction: When a hospital uploads patient records, the In-Enclave Engine processes the data within the secure enclave. It first constructs a lightweight inverted index (formatted as a delta-based structure) that maps key attributes (e.g., `patient_id`) to corresponding partition locations, similar to traditional DBMS indexing schemes but optimized for decentralized storage by supporting incremental updates. The raw data is encrypted using an enclave-generated symmetric key, partitioned into files, and persisted to IPFS. The index–storing key-value pairs of `patient_id` and CID for relevant partitions, and encrypted keys–is then published to IPFS, with its CID recorded on the blockchain for decentralized discovery. Detailed mechanisms for indexing are discussed in Section V-B.

2) Data Sharing with User Consent and Access Control: Since MtDB does not handle identity management internally, authorization interaction and access control policy determination should happen outside MtDB. The only requirements for MtDB are: 1. the finalized access policy from the specific owner; 2. a verification algorithm allowing the In-Enclave Engine to verify the owner's explicit consent through either one-time authorization (owner-issued certificate submitted with the query) or the querier's identity matching the access control policy stored on blockchain.

3) Policy-Enforced Query through Index: Upon receiving a query (e.g., `SELECT * FROM table WHERE name = ?`), the Query Interface routes it to the In-Enclave Engine. The enclave first verifies authorization using either patient-signed certificates or blockchain-stored access policies (check if there is explicit consent from the user with policies). It then dynamically rewrites the query to enforce patient-centric access control, appending ownership constraints (e.g., `AND patient_id = ?`). Specifically, the user's consent with the access policy acts like a first layer of filter to the database, which will be rewritten as a subquery over the original table to return the accessible part of the query request. Then the original query will be applied to the result of the subquery as the enforcement of the access control policy. Using the index's Content Identifier (CID) retrieved from the blockchain, the engine fetches the inverted index from IPFS. This index provides IPFS CIDs for all database partitions containing the target patient's id, and the corresponding symmetric keys for each partition decryption. The engine retrieves the encrypted partitions via IPFS, decrypts them within the enclave, and executes the rewritten query over the decrypted data. Results are returned through the Query Interface via HTTPS, ensuring end-to-end security where sensitive operations (authorization, decryption, query execution) occur exclusively within the TEE-protected environment.

D. Threat Model

We assume a powerful adversary with control over nearly all system components except the trusted hardware enclave (Intel SGX). The host operating system, hypervisor, privileged software, infrastructure providers, and system administrators are

considered untrusted and potentially compromised. The adversary may observe or manipulate memory outside the enclave, alter binaries, and intercept network traffic. Unauthorized users may attempt to access protected data through crafted queries or application-layer vulnerabilities. In the decentralized IPFS storage layer, some nodes may behave maliciously, but we assume the majority are honest and data retrieval by CID will eventually succeed through redundancy.

We assume Intel SGX guarantees confidentiality and integrity of code and data within enclaves: (i) the enclave isolation boundary is strictly hardware-enforced; and (ii) remote attestation reliably verifies enclave identity and integrity. Side-channel attacks and microarchitectural vulnerabilities are out of scope as they are orthogonal problems under active research, with Intel continuously mitigating them in newer SGX versions. These assumptions align with SGX architectural guarantees [16], [17] and follow established security assumptions of prior secure database systems [6], [9], [18]. All critical operations–access control enforcement, encryption/decryption, and query execution–are strictly confined to the SGX enclave to maintain end-to-end confidentiality.

V. Detailed Designs and Core Components

In this section, we discuss the core components to build MtDB, the logic behind each design choice, and potential optimization options.

A. Secure Metadata Sharing through Blockchain

The core idea of secure healthcare data sharing in MtDB is to use IPFS to handle the storage and the retrieval of actual data, while using the decentralized infrastructure of existing blockchain with TEE to share the metadata that is needed for data sharing management.

With each MtDB node equipped with its own Intel SGX enclaves hosting the engine, the distributed MtDB nodes can effectively form a secure and trustworthy network of TEEs over the blockchain by using confidential smart contracts. Specifically, the combination of blockchain and TEE nodes allows the user to store the ciphertext of private data on the blockchain, as in [29]–[31], while the corresponding symmetric key of the ciphertext can be shared among the TEE nodes (treated as a trusted party) through a simplified approach called distributed key generation (DKG). In this setup, the keys can be applied per contract, meaning all (registered) TEE nodes can access such keys, thereby supporting maintaining user-agnostic data in private.

B. Delta-based On-Chain Index Management

The index architecture in MtDB is designed to address the inherent challenges of decentralized, multi-tenant healthcare data management. Given that patient records are generated across hospitals, with the same `patient_id` potentially distributed across multiple IPFS partitions, the index serves two critical functions: 1) enabling efficient access control enforcement during queries, and 2) minimizing network I/O by reducing unnecessary partition retrievals. While `patient_id` is used as the default indexing attribute, the system supports dynamic attribute selection (e.g., unique identifiers such as `department_id` or `diagnosis_code`) to accommodate diverse application requirements.

Traditional synchronous index updates would incur prohibitive overhead due to three factors: First, the global index could grow exponentially with the number of entries in the dataset, making frequent full-index updates to IPFS impractical. Second, applying synchronous index updates in a distributed environment is highly impractical. Concurrent index modifications by multiple hospitals would require conflict-resolving mechanisms such as distributed consensus (e.g., CRDTs or blockchain-based ordering), introducing latency incompatible with real-time operations. Last but not least, storing the entire index as a single IPFS file would necessitate re-uploading the complete structure for every incremental change, amplifying network costs.

Algorithm 1: Delta-based index update (each node)

Input: Partitions $P = \{p_1, \ldots, p_n\}$, indexing attribute A_{idx} (e.g., `patient_id`), security parameter λ, shared enclave key k_{enclave}
Output: Delta CID Δ_c recorded on-chain

$\Delta \leftarrow$ new empty index
foreach $p_i \in P$ **do**
  $k_p \leftarrow \{0,1\}^\lambda$; $p_{\text{enc}} \leftarrow \mathsf{Enc}(p_i, k_{\text{p}})$
  $CID_p \leftarrow \mathsf{IPFS.Add}(p_{\text{enc}})$
  foreach $r \in p_i$ **do**
    $v \leftarrow r[A_{\text{idx}}]$
    if $v \notin \Delta$ **then**
      $\Delta[v] \leftarrow \emptyset$
    end
    $\Delta[v] \leftarrow \Delta[v] \cup \{(CID_p, k_{\text{p}})\}$
  end
end
$\Delta_{\text{enc}} \leftarrow \mathsf{Enc}(\Delta, k_{\text{enclave}})$
$\Delta_c \leftarrow \mathsf{IPFS.Add}(\Delta_{\text{enc}})$
$\mathsf{Blockchain.Record}(\Delta_c)$

1) Decentralized Delta-based Index Update: To resolve these issues, MtDB adopts an asynchronous delta-based update protocol (as shown in Algorithm 1), leveraging the append-only nature of healthcare data. When a hospital uploads new partitions:

- It generates lightweight index deltas–structured as key-value pairs mapping `patient_id`s to new partition CIDs and corresponding decryption keys.
- These deltas are encrypted and published to IPFS as standalone files.
- Only the delta CIDs are recorded on-chain via atomic transactions, creating an auditable trail of index changes.

2) Index Reconstruction Workflow: As shown in Algorithm 2, nodes dynamically reconstruct the latest index state during query processing by:

- Fetching all delta CIDs from the blockchain.

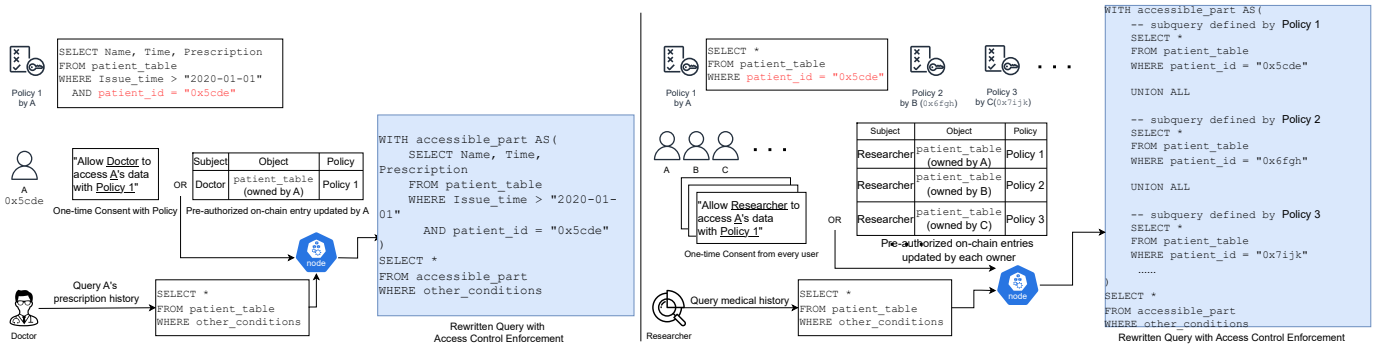


Fig. 3: The Query Rewriter in MtDB with example access control policy. The MtDB node will rewrite the query in the SGX enclave by replacing the table name in the original query with a subquery based on the access control policy (left). The policy can either be submitted with the one-time consent from the owner or be pre-defined and shared through the blockchain. When a query requires access to data with multiple owners (right), each data owner must provide explicit consent. The query will be rewritten with unions of all subqueries of policies.

- Retrieving and decrypting delta files from IPFS.
- Merging deltas into a consolidated in-enclave index.

$$\{key : \texttt{patient\_id}, value : [\texttt{CID} : \texttt{keys}]\}$$

This approach eliminates write conflicts by design–each delta is immutable and hospital-specific, while trading strict real-time consistency for substantial efficiency gains. The temporal gap between delta generation and consolidation is mitigated through periodic background synchronization, where nodes proactively cache recent deltas. Note that both algorithms operate entirely within the SGX enclave on each MtDB node. The encryption key k_{enclave} is securely generated inside the enclave and provisioned to authorized nodes via a remote attestation–based secure channel.

Algorithm 2: Index reconstruction from deltas

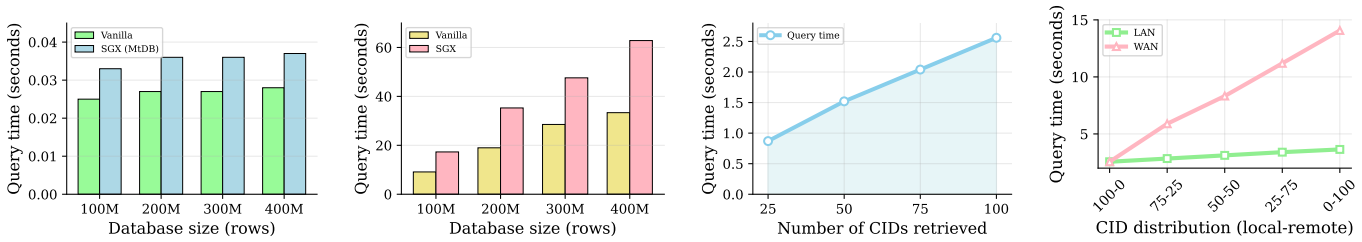
Input: Cached index I_{local}, last delta CID Δ_{last}, shared enclave key $k_{enclave}$
Output: Reconstructed global index I_{global}

$I_{\text{global}} \leftarrow I_{\text{local}}$
$\mathcal{C} \leftarrow \mathsf{Blockchain.GetDeltasSince}(\Delta_{\text{last}})$
foreach $\Delta_c \in \mathcal{C}$ **do**
  $\Delta_{\text{enc}} \leftarrow \mathsf{IPFS.Get}(\Delta_c)$
  $\Delta \leftarrow \mathsf{Dec}(\Delta_{\text{enc}}, k_{\text{enclave}})$
  foreach $v \in \Delta$ **do**
    if $v \notin I_{global}$ **then**
      $I_{\text{global}}[v] \leftarrow \emptyset$
    end
    $I_{\text{global}}[v] \leftarrow I_{\text{global}}[v] \cup \Delta[v]$
  end
end
return I_{global}

C. TEE-Enforced Owner-defined Access Control

MtDB deals with the multi-tenant table, which means the access control needs to be applied at the row record level. This idea is similar to the Row-Level Security concept applied by existing database systems [32], [33], in which the system administrator can set up row-level access policies against the queriers' properties (e.g., the role of the querier). Any query attempting to access row-level data in a table will be restricted by a security predicate defined as an inline table-valued function. The function will be invoked and enforced by the pre-defined security policy to filter the rows from the returned value of the query. The application that sends the query will be unaware of rows that are filtered from the result set. If all rows are filtered, then a null set is returned. MtDB achieves a fine-grained access control at the SQL statement level through a query rewriter. With the whole database engine inside the SGX enclave, MtDB is able to enforce row-level security in a multi-tenant table with additional security guarantees compared to the existing database systems. The security policy filters can be applied to the original query request and enforced with the integrity guarantee from the SGX enclave. Due to the memory isolation property of SGX, query execution remains entirely confidential, with no data leakage outside the enclave boundary.

In MtDB, we use a simple query-rewriter as a proof-of-concept (PoC) access control mechanism. Data owners can define the access policy of their data by writing a SQL statement that controls what can be queried. This SQL statement, which is the access policy, will be combined with the query to control which part of the data can be queried upon. An example is shown in Figure 3, where a policy together with other optional user-defined policies, such as granted views and filters, is applied when a querier submits a query. In the PoC implementation of MtDB, the original query will be rewritten by replacing the `table_name` with the access policy, which is a sub-query statement. This sub-query creates a view that limits the original query's searching domain and thus enforces the access control. MtDB rewriter is developed based on the widely used database query instrumentation techniques [34]–[36]. The rewritten query will then be processed by the query engine and executed over the table as a normal query process described in Section IV-C. The default ownership-



(a) With index (b) Without index (c) Scalability vs CIDs (d) Network distribution

Fig. 4: Query efficiency of MtDB under varying configurations. (a) Query latency with index remains nearly constant across database sizes. (b) Without index, latency increases linearly. (c) Query time scales linearly with the number of matched partitions (CIDs). (d) Network distribution analysis shows latency increases with remote data retrieval, especially over WAN.

based filtering also allows MtDB to utilize the ownership-based index from Section V-B to reduce the number of CIDs that need to be retrieved from IPFS.

Note that the access control enforcement can also be achieved through other DBMS functionalities, like inline table-value functions in Microsoft SQL Server [32], as long as the whole engine is under the shelter of the SGX enclave. However, they are less flexible than MtDB's query rewriter and may incur additional computational and space cost as the access policies change. The query-rewriting-based solution is simply a choice that is agnostic to any SQL-like database engines, which enforces the access control policy directly on the SQL statement. The rewritten query can work with any SQL-based engine. Such an approach also benefits from the query optimizer of the existing database engines, which finds the cheapest query plan to be executed by rearranging operators in the query.

VI. Evaluation

We implemented and evaluated MtDB in two configurations: 1) within an SGX enclave to assess performance under secure execution, and 2) on the host operating system as a non-secure baseline. Each experiment was executed five times, and we report the median to mitigate outliers.

A. Experimental Setup

Our prototype integrates DuckDB within a Gramine SGX enclave. Both SGX-secure and non-secure (vanilla) variants were deployed on Ubuntu 22.04.5 LTS with 503 GB RAM, 868 GB SSD, and Intel Xeon Gold 5412U with SGX Version 2 support. We used Gramine 1.8 (a library OS to run the binaries) and increased the Enclave Page Cache size to 128 GB leveraging SGX v2 capabilities. We evaluated MtDB using a synthetic healthcare dataset of 400 million records (42.4 GB) partitioned into 4,000 IPFS-addressable chunks. Each chunk contains 100,000 realistic medical records.

B. Query Execution Time

1) Index Performance: We executed point queries (`SELECT * FROM table WHERE patient_id = X`) across datasets from 100M to 400M rows. As shown in Figure 4a, indexing significantly reduces query latency, maintaining nearly constant performance across database sizes. Without indexing (Figure 4b), execution time increases linearly due to full partition scanning. At 400M rows, SGX takes 62.8 seconds without indexing versus 37 milliseconds with indexing–a 1700× improvement. The SGX-secure variant incurs modest 1.2–1.3× overhead compared to vanilla baseline (37 ms vs. 28 ms at 400M rows) due to SGX architectural bottlenecks and Gramine abstraction.

2) Scalability with Number of CIDs: We measured query performance versus CIDs retrieved using range queries (`SELECT * FROM table WHERE age > X`). Figure 4c shows linear scaling from 0.87 seconds (25 CIDs) to 2.56 seconds (100 CIDs). MtDB maintains low latency even at upper bounds, demonstrating efficient multi-partition query handling.

3) Impact of Network Distribution: We evaluated query latency with data distributed across local and remote hosts via LAN/WAN. Figure 4d shows modest LAN impact (42% increase) but substantial WAN overhead (5.5× slowdown). WAN latency is calculated as:

$$T_{\text{fetch}} = \frac{N \cdot S_{\text{CID}}}{B} + \delta, \tag{1}$$

where $S_{\text{CID}} = 11.848 \times 10^6$ bits, $B = 100 \times 10^6$ bits/sec, and $\delta = 70 \times 10^{-3}$ sec for coast-to-coast latency.

C. Index Size and Maintenance Overhead

TABLE I: Index Size vs. Number of Rows

Total Number of Rows	Encrypted Index Size (MB)
100M	41.52
200M	89.79
300M	186.34
400M	379.44

Table I shows near-linear index growth requiring 0.95 MB per million records. The encrypted index is stored on IPFS with CID registered on Ethereum blockchain. MtDB employs delta-based updates appending only incremental changes, ensuring constant-time updates and supporting logarithmic-time lookups through our custom tree structure index.

VII. Conclusion

In this paper, we propose MtDB, a decentralized multi-tenant database architecture for secure and ownership-centric

data sharing in healthcare ecosystems. MtDB utilizes existing decentralized infrastructures: IPFS for content-addressable storage and blockchain for tamper-proof metadata sharing. Each node uses a TEE to achieve secure encrypted metadata sharing. With TEE's remote attestation and memory isolation, MtDB enforces user queries with access control policy and guarantees execution integrity. The modular design coordinates between components, allowing different query engines for various application scenarios.

ACKNOWLEDGMENT

This work is partially sponsored by NASA ULI under Grant No. 80NSSC23M0058, and NSF under Grant No. OAC-2312973.

REFERENCES

[1] "General Data Protection Regulation (GDPR) – Official Legal Text — gdpr-info.eu," https://gdpr-info.eu/, [Accessed: Aug. 2, 2025].

[2] "Health Insurance Portability and Accountability Act (HIPAA) Compliance — ncbi.nlm.nih.gov," [Accessed: Aug. 2, 2025].

[3] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss *et al.*, "Cockroachdb: The resilient geo-distributed sql database," in *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, 2020, pp. 1493–1509.

[4] Apache Software Foundation, "Apache Cassandra," https://cassandra.apache.org/, 2025, [Accessed: Aug. 2, 2025].

[5] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan *et al.*, "Spanner: Becoming a sql system," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 331–343.

[6] X. Yang, C. Yue, W. Zhang, Y. Liu, B. C. Ooi, and J. Chen, "Secudb: An in-enclave privacy-preserving and tamper-resistant relational database," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 3906–3919, 2024.

[7] S. Wang, Y. Li, H. Li, F. Li, C. Tian, L. Su, Y. Zhang, Y. Ma, L. Yan, Y. Sun *et al.*, "Operon: An encrypted database for ownership-preserving data management," *Proceedings of the VLDB Endowment*, vol. 15, no. 12, pp. 3332–3345, 2022.

[8] Y. Chen, Q. Zheng, Z. Yan, and D. Liu, "Qshield: Protecting outsourced cloud data queries with multi-user access control based on sgx," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 485–499, 2020.

[9] C. Priebe, K. Vaswani, and M. Costa, "Enclavedb: A secure database using sgx," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 264–278.

[10] S. S. Mukherjee, W. Tang, G. P. F. Aniceto, J. Chandler, W. Song, and T. Jung, "Web3db: Web 3.0 rdbms for individual data ownership," *arXiv preprint arXiv:2504.02713*, 2025.

[11] A. Haddad, M. H. Habaebi, E. A. Elsheikh, M. R. Islam, S. A. Zabidi, and F. E. M. Suliman, "E2ee enhanced patient-centric blockchain-based system for ehr management," *Plos one*, vol. 19, 2024.

[12] M. Lücking, R. Manke, M. Schinle, L. Kohout, S. Nickel, and W. Stork, "Decentralized patient-centric data management for sharing iot data streams," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2020, pp. 1–6.

[13] J. Li and J. Meng, "Ipfs-dkrm: An efficient keyword retrieval model of ipfs based on art," in *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 2024, pp. 123–135.

[14] M. M. Arer, P. M. Dhulavvagol, and S. Totad, "Efficient big data storage and retrieval in distributed architecture using blockchain and ipfs," in *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*. IEEE, 2022, pp. 1–6.

[15] T. Rathee and M. Malik, "Blockchain as an ipfs (interplanetary file system) storage index," *Int. J. Innovations Eng. Technol.(IJIET)*, vol. 14, no. 2, pp. 1–6, 2019.

[16] V. Costan and S. Devadas, "Intel sgx explained," *Cryptology ePrint Archive*, 2016.

[17] M. El-Hindi, T. Ziegler, M. Heinrich, A. Lutsch, Z. Zhao, and C. Binnig, "Benchmarking the second generation of intel sgx hardware," in *Proceedings of the 18th International Workshop on Data Management on New Hardware*, 2022, pp. 1–8.

[18] D. Vinayagamurthy, A. Gribov, and S. Gorbunov, "Stealthdb: a scalable encrypted database with full sql query support," *Proceedings on Privacy Enhancing Technologies*, 2019.

[19] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 85–100.

[20] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Opaque: An oblivious and encrypted distributed analytics platform," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 283–298.

[21] G. Zyskind, O. Nathan, and A. Pentland, "Enigma: Decentralized computation platform with guaranteed privacy," *arXiv preprint arXiv:1506.03471*, 2015.

[22] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, "Bigchaindb: a scalable blockchain database," *white paper, BigChainDB*, pp. 53–72, 2016.

[23] OrbitDB Contributors, "Orbitdb: Peer-to-peer database for the decentralized web," https://orbitdb.org/, 2023, [Accessed: Aug. 2, 2025].

[24] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1594–1605, 2018.

[25] C. Liu, M. Xu, H. Guo, X. Cheng, Y. Xiao, D. Yu, B. Gong, A. Yerukhimovich, S. Wang, and W. Lyu, "Tbac: A tokoin-based accountable access control scheme for the internet of things," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 6133–6148, 2023.

[26] P. Wang, N. Xu, H. Zhang, W. Sun, and A. Benslimane, "Dynamic access control and trust management for blockchain-empowered iot," *IEEE Internet of Things Journal*, vol. 9, no. 15, 2021.

[27] "Verifiable Credentials Data Model v1.1." [Online]. Available: https://www.w3.org/TR/vc-data-model/#abstract

[28] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 790–808.

[29] M. Bowman, A. Miele, M. Steiner, and B. Vavala, "Private data objects: an overview," *arXiv preprint arXiv:1807.05686*, 2018.

[30] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie, "Shadoweth: Private smart contract on public blockchain," *Journal of Computer Science and Technology*, vol. 33, pp. 542–556, 2018.

[31] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200.

[32] Microsoft Documentation, *Row-Level Security*, 2023. [Online]. Available: https://learn.microsoft.com/en-us/sql/relational-databases/security/row-level-security?view=sql-server-ver17

[33] PostgreSQL Global Development Group, *Row Security Policies*, 2023. [Online]. Available: https://www.postgresql.org/docs/current/ddl-rowsecurity.html

[34] B. Glavic and G. Alonso, "Perm: Processing Provenance and Data on the same Data Model through Query Rewriting," in *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE)*, 2009, pp. 174–185.

[35] S. Lee, B. Ludäscher, and B. Glavic, "Pug: a framework and practical implementation for why and why-not provenance," *The VLDB Journal*, pp. 1–25, 2018.

[36] J. Turnau, N. Akwari, S. Lee, and D. Rajput, "Provenance-based explanations for machine learning (ml) models," in *2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 2023, pp. 40–43.