MULTITENANT RESEARCH WORK

1. Introduction

Software as a Service (SaaS) commonly employs a multi-tenant approach, where users and their end-users share software and hardware resources without necessarily sharing data. Surprisingly, there is little consensus on the definition, scope, and challenges of multi-tenancy, impeding progress in research and industry. This research presents a systematic scalable solution for a betterment of tenancy transaction with Blockchain Technology. The research projects and proofs how Datas can be securely transferred using BlockChain Technology and also proves the possibility of using Smart Contracts, Dynamic Network Creations and New Methodology for Encryptions of Chunks in RAID 10 Data Storage.

2. Research Method

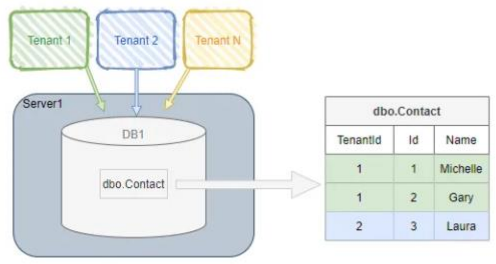
Our research adopts a systematic approach to address the identified challenges. We analyze and integrate insights from existing academic papers, industrial blogs, and related research initiatives. The proposed solution involves the utilization of Blockchain Technology to enhance the security and efficiency of multi-tenancy transactions.

2.1 Network Creation

The initial phase involves the conceptualization of a generic network framework that is adaptable to diverse domains. This design is informed by principles of scalability, flexibility, and security.

2.1.1 Existing Networks

2.1.2. Single database, shared schema :



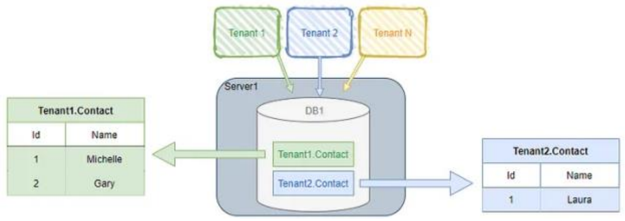
dbo.Contact

TenantId	Id	Name
1	1	Michelle
1	2	Gary
2	3	Laura

Problem Faced

- Single database schema to maintain and a simple schema update rollout process so that the process needs to be applied once.
- Manage the High Availability/ Disaster Recovery/ Maintenance operation/ Monitoring strategy for a single database.
- Single schema, Single database to connect.

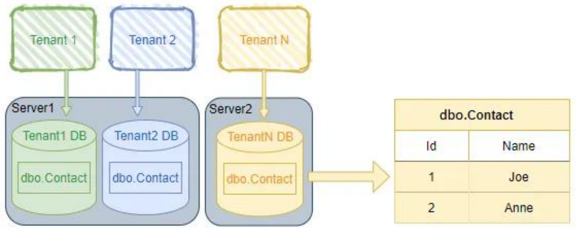
2.1.3 Single database, Separate schema :



Problems Faced

- Can’t easily restore a single tenant’s data (although it’s a slightly better process than (Single database, shared schema approach) due to isolation of tenant data).
- As the number of tenants grows, there will be a lot of database objects being created to manage and maintain.
- Schema updates are more involved, needing to be rolled out to n tenants.

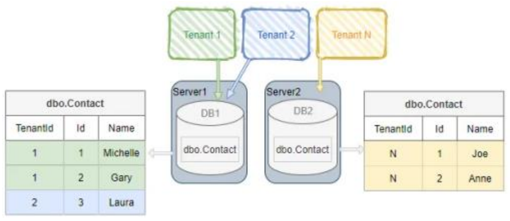
2.1.4 Database Per Tenant :



Problem Faced

- Potentially more servers to patch and keep secure.
- As the number of tenants grows, there will be a lot of database objects being created to manage and maintain.
- Adding new tenants is more involved, as new schemas need to be created.

2.1.5 Multiple Databases, Multiple Tenants Per Database, Shared Schema



Problem Faced

- Tenants still share a database and schema with others.
- More maintenance is required.

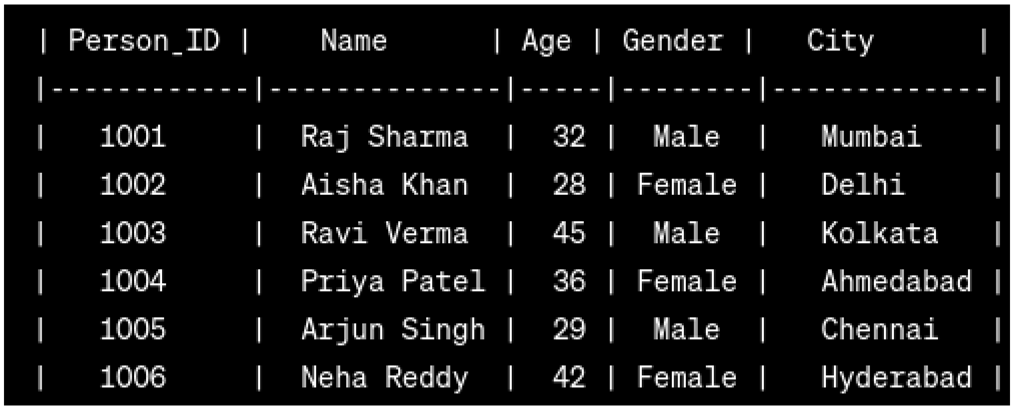
Proposed system

In response to the challenges posed by the existing multi-tenancy database architectures, we propose a revolutionary approach that leverages dynamic topology of tenant connectivity and integrates blockchain technology to overcome the identified problems. By adopting a dynamic topology, we aim to create a flexible and scalable environment where tenants can seamlessly connect without the constraints of a fixed schema. This dynamic connectivity allows for enhanced data isolation, enabling efficient restoration of individual tenant data without the complexities associated with a single shared schema. Additionally, we introduce blockchain technology to manage the connections between tenants securely. The decentralized and tamper-resistant nature of blockchain ensures the integrity and security of tenant connections, addressing concerns related to High Availability, Disaster Recovery, and Monitoring strategies. This innovative approach minimizes the need for extensive maintenance and schema updates, providing a robust foundation for a scalable, secure, and easily manageable multi-tenancy database architecture. By decentralizing the connectivity and securing it with blockchain, we mitigate the challenges posed by the traditional models, offering a transformative solution for a dynamic and secure multi-tenant database ecosystem.

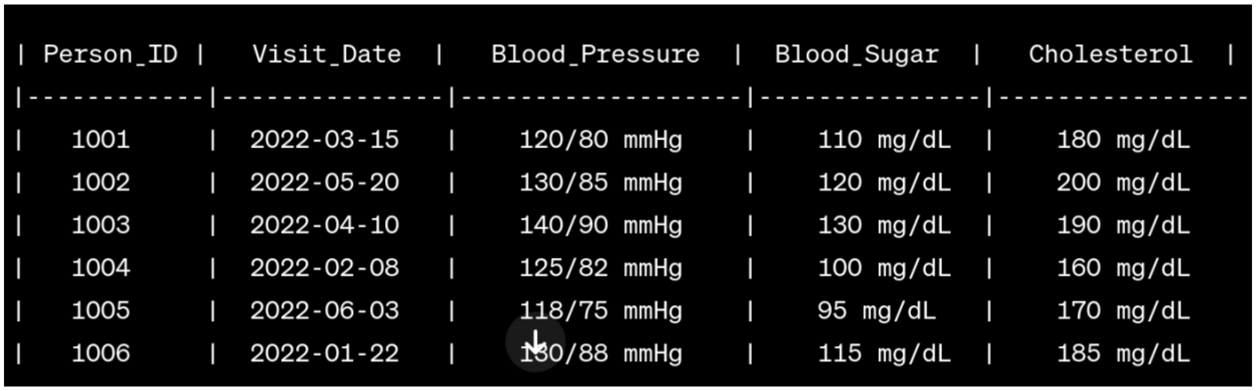
Objectives

Secured Data Transaction of Tenants using Blockchain Technology:

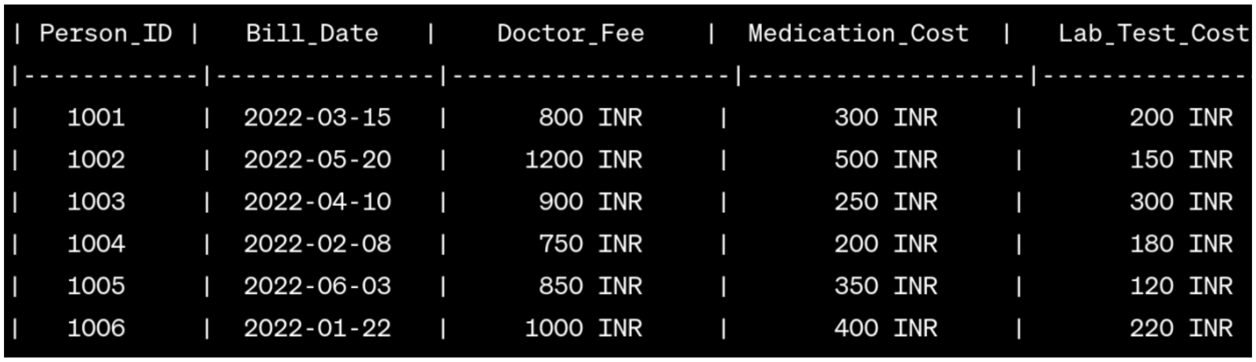
In our proposed solution, we integrate Blockchain Technology to ensure the secured transaction of data among tenants. Blockchain, being a decentralized and tamper-resistant ledger, provides an immutable record of all transactions. Each data transaction is cryptographically secured, ensuring data integrity and preventing unauthorized access. This not only enhances the overall security of tenant data but also establishes a transparent and accountable system for data transactions within the multi-tenant environment.



```
| Person_ID |     Name      | Age | Gender |    City      |
|-----------|---------------|-----|--------|--------------|
|   1001    |  Raj Sharma   |  32 |  Male  |   Mumbai     |
|   1002    |  Aisha Khan   |  28 | Female |   Delhi      |
|   1003    |  Ravi Verma   |  45 |  Male  |   Kolkata    |
|   1004    |  Priya Patel  |  36 | Female |   Ahmedabad  |
|   1005    |  Arjun Singh  |  29 |  Male  |   Chennai    |
|   1006    |  Neha Reddy   |  42 | Female |   Hyderabad  |
```



```
| Person_ID |   Visit_Date   |   Blood_Pressure   |  Blood_Sugar   |   Cholesterol   |
|-----------|----------------|--------------------|----------------|-----------------
|   1001    |  2022-03-15    |    120/80 mmHg     |    110 mg/dL   |    180 mg/dL
|   1002    |  2022-05-20    |    130/85 mmHg     |    120 mg/dL   |    200 mg/dL
|   1003    |  2022-04-10    |    140/90 mmHg     |    130 mg/dL   |    190 mg/dL
|   1004    |  2022-02-08    |    125/82 mmHg     |    100 mg/dL   |    160 mg/dL
|   1005    |  2022-06-03    |    118/75 mmHg     |    95 mg/dL    |    170 mg/dL
|   1006    |  2022-01-22    |    130/88 mmHg     |    115 mg/dL   |    185 mg/dL
```



```
| Person_ID |   Bill_Date    |     Doctor_Fee     |  Medication_Cost   |   Lab_Test_Cost
|-----------|----------------|--------------------|--------------------|--------------
|   1001    |  2022-03-15    |       800 INR      |       300 INR      |       200 INR
|   1002    |  2022-05-20    |      1200 INR      |       500 INR      |       150 INR
|   1003    |  2022-04-10    |       900 INR      |       250 INR      |       300 INR
|   1004    |  2022-02-08    |       750 INR      |       200 INR      |       180 INR
|   1005    |  2022-06-03    |       850 INR      |       350 INR      |       120 INR
|   1006    |  2022-01-22    |      1000 INR      |       400 INR      |       220 INR
```

- **`Person_ID`: A unique identifier for each individual, linking the medical report and the bill datasets.**
- **`Visit_Date` (in the medical report dataset): The date of the medical visit.**
- **`Blood_Pressure`, `Blood_Sugar`, `Cholesterol`: Different health parameters recorded during the medical visit.**
- **`Diagnosis`: The medical condition diagnosed during the visit.**
- **`Bill_Date` (in the medical bill dataset): The date of the billing.**
- **`Doctor_Fee`, `Medication_Cost`, `Lab_Test_Cost`: The costs associated with the medical visit, including doctor's fees, medication, and lab tests.**
- **`Total_Bill`: The total cost incurred during the visit.**

Mathematical Framework for Blockchain-based Data Security:

Cryptographic Hash Functions:

H(x)=YH(x)=Y

Where H(x)H(x) represents the cryptographic hash function, xx denotes the input data, and YY is the resulting hash value. This equation illustrates the use of cryptographic hashing to secure data integrity within the blockchain.

Digital Signatures:

Sign Private key(x)=SignatureSignprivate key(x)=Signature

The equation depicts the process of generating a digital signature using a private key, ensuring data authenticity and non-repudiation within the blockchain.

Transaction Verification:

Verifypublickey(Signature,Data,Blockchain)={True, False}Verifypublic key (Signature,Data,Blockchain)={True, False}

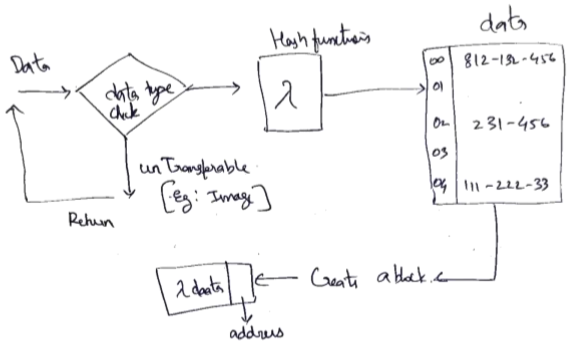
This equation represents the verification process, ensuring that data transactions are authentic and valid through the use of public key verification.

Application to Tenant Data Transactions:

Secure Data Transaction:

Transactionsecure={Data,Hash,Signature}

This equation encapsulates a secure data transaction, including the data itself, its cryptographic hash, and the digital signature, ensuring both integrity and authenticity.



The mathematical representation of the hash function with both data, key, and address can be expressed as:

Hash_Key=H(Data,Key,Address)

For the SHA-256 algorithm, this can be further detailed as:

Hash_Key=SHA-256(Data,Key,Address)=SHA-256(Data[0],Data[1],…,Data[n−1],Key,Address)

Here, Data[i]Data[i] represents the ii-th block of the input data, KeyKey is the secret key, and AddressAddress is an additional parameter representing an address.

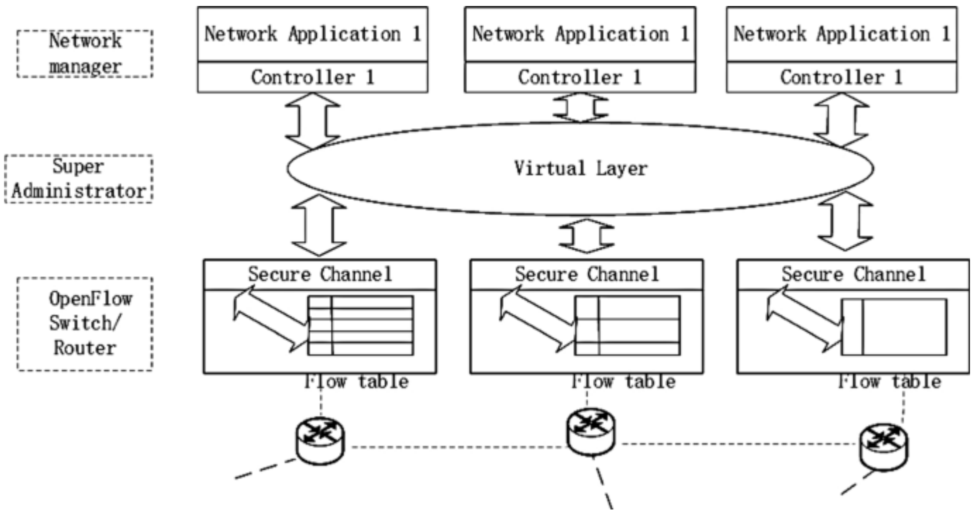
The specific mathematical operations involving the key, address, and data will depend on the design of the hash function. In many cryptographic hash functions, these parameters are mixed using operations such as XOR (⊕⊕), modular addition (++), or bitwise rotation to ensure the uniqueness and security of the resulting hash value.

For example, a simplified representation of introducing both a key (KeyKey) and an address (AddressAddress) into the hashing process might be:

Hash_Key=SHA-256(Data,Key,Address)=SHA-256(Data⊕Key⊕Address)Hash_Key=SHA-256(Data,Key,Address)=SHA-256(Data⊕Key⊕Address)

In this representation, the data, key, and address are combined using XOR operations before applying the SHA-256 algorithm. This ensures that changes in any of the parameters result in a significantly different hash value. Introducing both a key and an address enhances the security and complexity of the hash function, making it more resistant to various cryptographic attacks.

Decentralized Data Ledger:



Blockchaintenant={Block1,Block2,…,Blockn}

The blockchain for each tenant is mathematically represented as a series of blocks, each containing securely hashed and signed data transactions, forming a decentralized and tamper-resistant ledger.

Here comes the research that is done in this objective below

NetworkTenant=f(Connectivity,Topology,Join/Leave)

Where:

- NetworkTenantNetworkTenant is the decentralized network for tenants.
- ff is a function representing the dynamics of the network.
- ConnectivityConnectivity denotes the degree and quality of connectivity between tenants in the network.
- TopologyTopology refers to the structure or arrangement of the network.
- Join/LeaveJoin/Leave represents the ability of tenants to join or leave the network dynamically.
- **Initialization of Ledger:** LedgerChain=AM·Genesis_BlockLedgerChain

  =AM·Genesis_Block
- **Adding a New Block to the Ledger:**

  LedgerChain=AM·Add_Block(LedgerChain,AM·Blocki)LedgerChain

  =AM·Add_Block(LedgerChain,AM·Blocki)

  Blocki=AM·Mine_Block(Transactionsi,AM·Previous_Hashi−1,AM·Noncei)Blocki

  =AM·Mine_Block(Transactionsi,AM·Previous_Hashi−1,AM·Noncei)

  LedgerChain=AM·Validate_and_Add(LedgerChain,AM·Blocki)LedgerChain

  =AM·Validate_and_Add(LedgerChain,AM·Blocki)
- **Transaction Verification:**

  Transactionj=AM·Verify_Transaction(Transactionj,AM·Public_KeySender,AM·SignatureS

  ender)Transactionj=AM·Verify_Transaction(Transactionj,AM·Public_KeySender

  ,AM·SignatureSender)
- **Consensus Mechanism:**

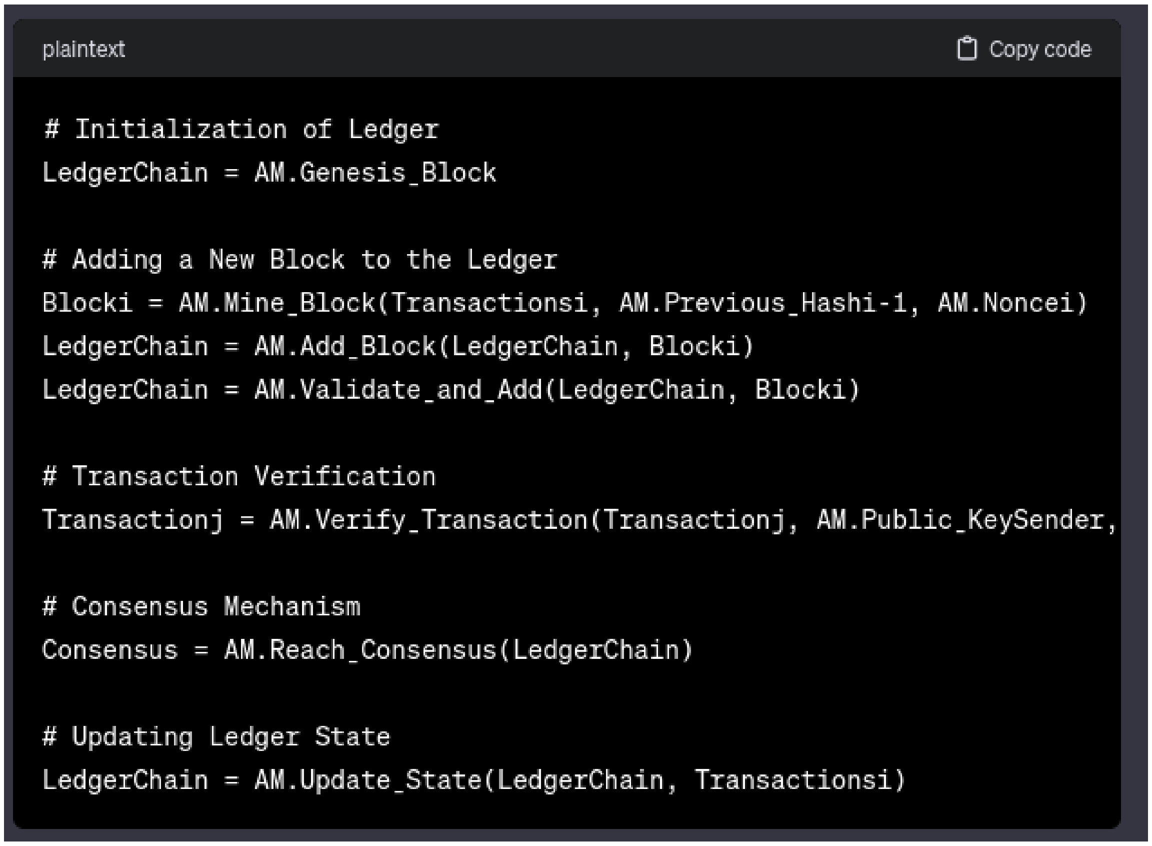
  Consensus=AM·Reach_Consensus(LedgerChain)Consensus=AM·Reach_Consensus(L

  edgerChain)
- **Updating Ledger State:**

  LedgerChain=AM·Update_State(LedgerChain,AM·Transactionsi)LedgerChain

  =AM·Update_State(LedgerChain,AM·Transactionsi)

Code:



plaintext

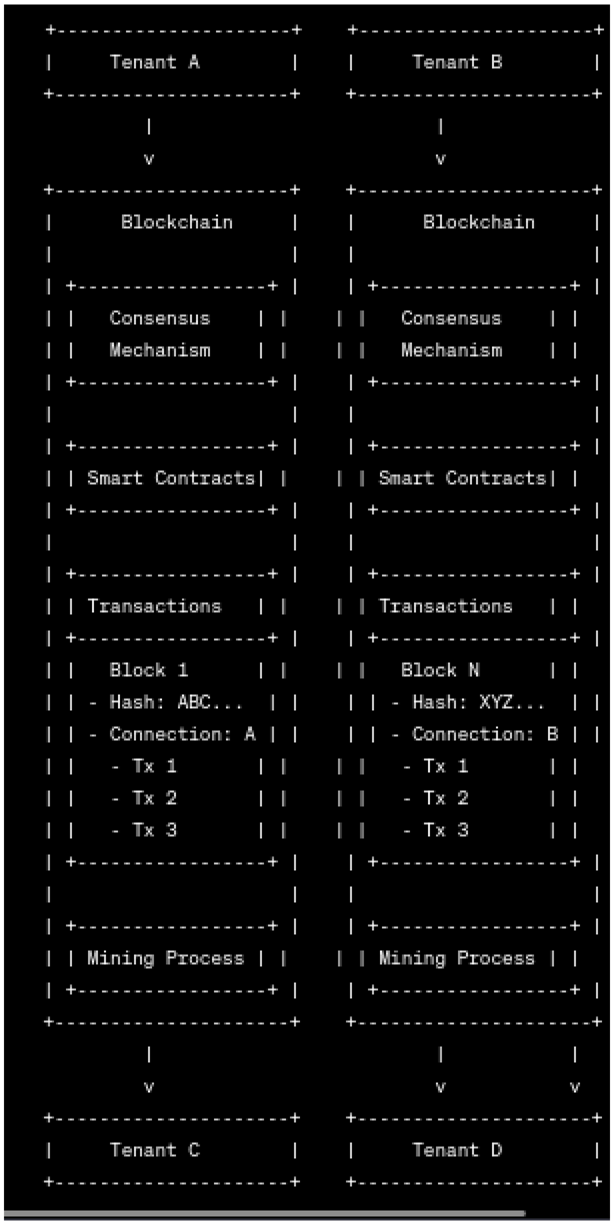
Copy code

```
# Initialization of Ledger
LedgerChain = AM.Genesis_Block

# Adding a New Block to the Ledger
Blocki = AM.Mine_Block(Transactionsi, AM.Previous_Hashi-1, AM.Noncei)
LedgerChain = AM.Add_Block(LedgerChain, Blocki)
LedgerChain = AM.Validate_and_Add(LedgerChain, Blocki)

# Transaction Verification
Transactionj = AM.Verify_Transaction(Transactionj, AM.Public_KeySender,

# Consensus Mechanism
Consensus = AM.Reach_Consensus(LedgerChain)

# Updating Ledger State
LedgerChain = AM.Update_State(LedgerChain, Transactionsi)
```

This equation encapsulates the essential components of a decentralized network for tenants. The function ff could involve specific algorithms, protocols, or rules that govern how tenants connect, the overall network structure, and the mechanisms for tenants to join or leave the

network seamlessly.

Tennant Creation - Architecture Diagram

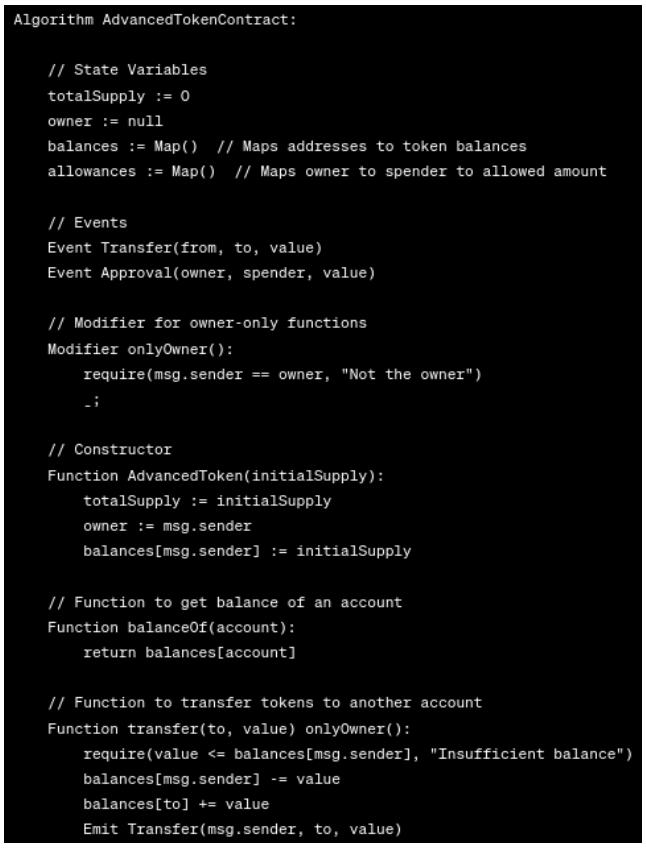


```
+--------------------+    +--------------------+
|     Tenant A       |    |     Tenant B       |
+--------------------+    +--------------------+
         |                         |
         v                         v
+--------------------+    +--------------------+
|      Blockchain    |    |      Blockchain    |
|                    |    |                    |
| +----------------+ |    | +----------------+ |
| |   Consensus    | |   | |   Consensus    | |
| |   Mechanism    | |   | |   Mechanism    | |
| +----------------+ |    | +----------------+ |
|                    |    |                    |
| +----------------+ |    | +----------------+ |
| | Smart Contracts| |   | | Smart Contracts| |
| +----------------+ |    | +----------------+ |
|                    |    |                    |
| +----------------+ |    | +----------------+ |
| | Transactions   | |   | | Transactions   | |
| +----------------+ |    | +----------------+ |
| |   Block 1      | |   | |   Block N      | |
| | - Hash: ABC...  | |    | | - Hash: XYZ...  | |
| | - Connection: A | |    | | - Connection: B | |
| |   - Tx 1       | |   | |   - Tx 1       | |
| |   - Tx 2       | |   | |   - Tx 2       | |
| |   - Tx 3       | |   | |   - Tx 3       | |
| +----------------+ |    | +----------------+ |
|                    |    |                    |
| +----------------+ |    | +----------------+ |
| | Mining Process | |   | | Mining Process | |
| +----------------+ |    | +----------------+ |
+--------------------+    +--------------------+
         |                         |          |
         v                         v          v
+--------------------+    +--------------------+
|     Tenant C       |    |     Tenant D       |
+--------------------+    +--------------------+
```

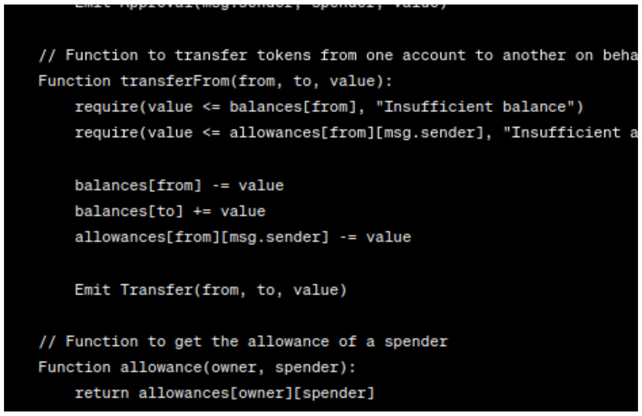
- Each block in the blockchain now includes a hash value (e.g., ABC...) and a connection address (e.g., A or B) for illustrative purposes.
- The hash represents the unique identifier for the block, ensuring its integrity.
- The connection address indicates the source or owner of the block.

Advanced Smart Contract Implementations for Transactions:

Smart Contracts play a pivotal role in automating and enforcing the terms of agreements between tenants. Our solution incorporates advanced Smart Contract implementations to streamline and automate complex transaction processes. These Smart Contracts execute predefined rules and conditions, ensuring that data transactions occur seamlessly, securely, and in accordance with the agreed-upon terms. This not only reduces the need for manual intervention but also enhances the efficiency and reliability of transactions within the multi-tenant system.



```
Algorithm AdvancedTokenContract:

    // State Variables
    totalSupply := 0
    owner := null
    balances := Map()  // Maps addresses to token balances
    allowances := Map()  // Maps owner to spender to allowed amount

    // Events
    Event Transfer(from, to, value)
    Event Approval(owner, spender, value)

    // Modifier for owner-only functions
    Modifier onlyOwner():
        require(msg.sender == owner, "Not the owner")
        _;

    // Constructor
    Function AdvancedToken(initialSupply):
        totalSupply := initialSupply
        owner := msg.sender
        balances[msg.sender] := initialSupply

    // Function to get balance of an account
    Function balanceOf(account):
        return balances[account]

    // Function to transfer tokens to another account
    Function transfer(to, value) onlyOwner():
        require(value <= balances[msg.sender], "Insufficient balance")
        balances[msg.sender] -= value
        balances[to] += value
        Emit Transfer(msg.sender, to, value)
```



```
        Emit Approval(msg.sender, spender, value)

    // Function to transfer tokens from one account to another on beha
    Function transferFrom(from, to, value):
        require(value <= balances[from], "Insufficient balance")
        require(value <= allowances[from][msg.sender], "Insufficient a

        balances[from] -= value
        balances[to] += value
        allowances[from][msg.sender] -= value

        Emit Transfer(from, to, value)

    // Function to get the allowance of a spender
    Function allowance(owner, spender):
        return allowances[owner][spender]
```

1. **State Variables:**
    - **`totalSupply`: Represents the total supply of tokens in the contract.**
    - **`owner`: Stores the address of the contract owner.**
    - **`balances`: A mapping that associates addresses with their corresponding token balances.**
    - **`allowances`: A mapping that keeps track of the approved allowances for spending tokens by one address on behalf of another.**

2. **Events:**
    - `Transfer(from, to, value)`: **An event emitted when tokens are transferred from one address to another.**
    - `Approval(owner, spender, value)`: **An event emitted when the owner approves a spender to spend a certain amount of tokens.**
3. **Modifier:**
    - `onlyOwner()`: **A modifier ensuring that only the owner of the contract can execute certain functions. It checks whether the sender of the transaction is the owner.**
4. **Constructor:**
    - `AdvancedToken(initialSupply)`: **Initializes the contract with an initial token supply. It sets the total supply, designates the contract creator as the owner, and allocates the entire supply to the owner's address.**
5. **Functions:**
    - `balanceOf(account)`: **Retrieves the token balance of a specified account.**
    - `transfer(to, value)`: **Allows the owner to transfer tokens to another address. It checks for a sufficient balance and emits a** `Transfer` **event upon successful transfer.**
    - `approve(spender, value)`: **Permits the owner to approve a spender to spend a specified amount of tokens on their behalf. It emits an** `Approval` **event.**
    - `transferFrom(from, to, value)`: **Allows a designated spender to transfer tokens from one address to another on behalf of the owner, considering the approved allowance. It emits a** `Transfer` **event.**
    - `allowance(owner, spender)`: **Retrieves the approved allowance of a spender to spend tokens on behalf of an owner.**

Dynamic Network Creation:

To address the challenges associated with fixed network structures, we propose the implementation of a Dynamic Network Creation mechanism. This allows tenants to dynamically connect and disconnect from the network as needed, providing flexibility and scalability. The dynamic network creation ensures that tenants have the ability to adapt to changing requirements, enabling a more agile and responsive multi-tenant environment. This approach mitigates the issues related to rigid network configurations, making the system more adaptable to the evolving needs of tenants. The equations is explained under data ledger section above.

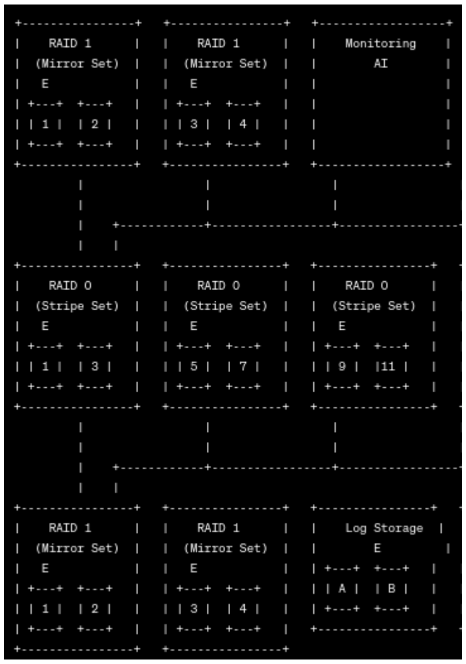
Coin Transaction & Protocols of Key Generation:

To enhance the security and privacy of data transactions, our solution incorporates coin transactions and protocols for key generation. Each transaction is associated with a cryptographic coin, and the key generation protocols ensure secure communication between tenants. This not only adds an additional layer of security but also facilitates traceability and accountability in data transactions. The use of cryptographic coins and robust key generation protocols strengthens the overall security posture of the multi-tenant system.

Chunks with RAID Concept Encryption:

In addressing the challenges of managing and maintaining a large number of database objects, our solution employs a Chunks with RAID (Redundant Array of Independent Disks) Concept Encryption. This approach optimizes storage efficiency while providing fault tolerance. Data is fragmented into chunks, and the RAID concept ensures redundancy and reliability. Each chunk is individually encrypted, enhancing data security. This method not only simplifies the management of a growing number of tenants but also ensures the integrity and availability of data through redundancy and encryption.

In summary, our comprehensive solution leverages Blockchain Technology, advanced Smart Contracts, dynamic network creation, coin transactions with key generation protocols, and chunks with RAID concept encryption to revolutionize the multi-tenant database architecture. This integrated approach addresses security, scalability, and flexibility concerns, providing a robust foundation for a modern and efficient multi-tenant system.



1. **RAID 1 (Mirror Sets):**
   - There are multiple RAID 1 (mirror) sets represented by the first and second rows.
   - Each RAID 1 set consists of two drives that mirror each other, providing redundancy.
   - The "E" in each RAID 1 block represents encryption, indicating that the data stored on these mirrored sets is encrypted.
2. **RAID 0 (Stripe Sets):**

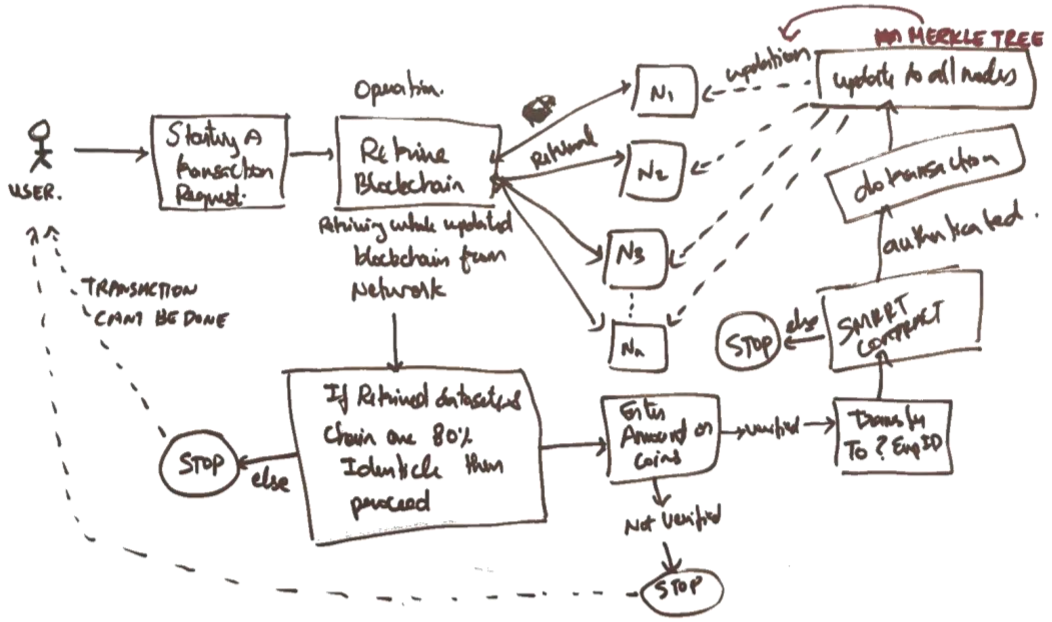
    - The third row consists of RAID 0 (stripe) sets, where data is striped across the RAID 1 sets for improved performance.
    - Each RAID 0 set consists of two drives, and the "E" indicates encryption for the striped data.
3. **Monitoring AI:**
    - The central "Monitoring AI" block represents an artificial intelligence system monitoring the entire RAID configuration.
    - This AI system is responsible for overseeing the health, performance, and security of the storage system.
4. **Log Storage:**
    - Two "Log Storage" blocks are included in the last row, each with its own set of mirrored drives.
    - The "E" in each log storage block indicates encryption for the stored log data.
    - Log storage is crucial for recording events, errors, and activities within the storage system, aiding in diagnostics and troubleshooting.

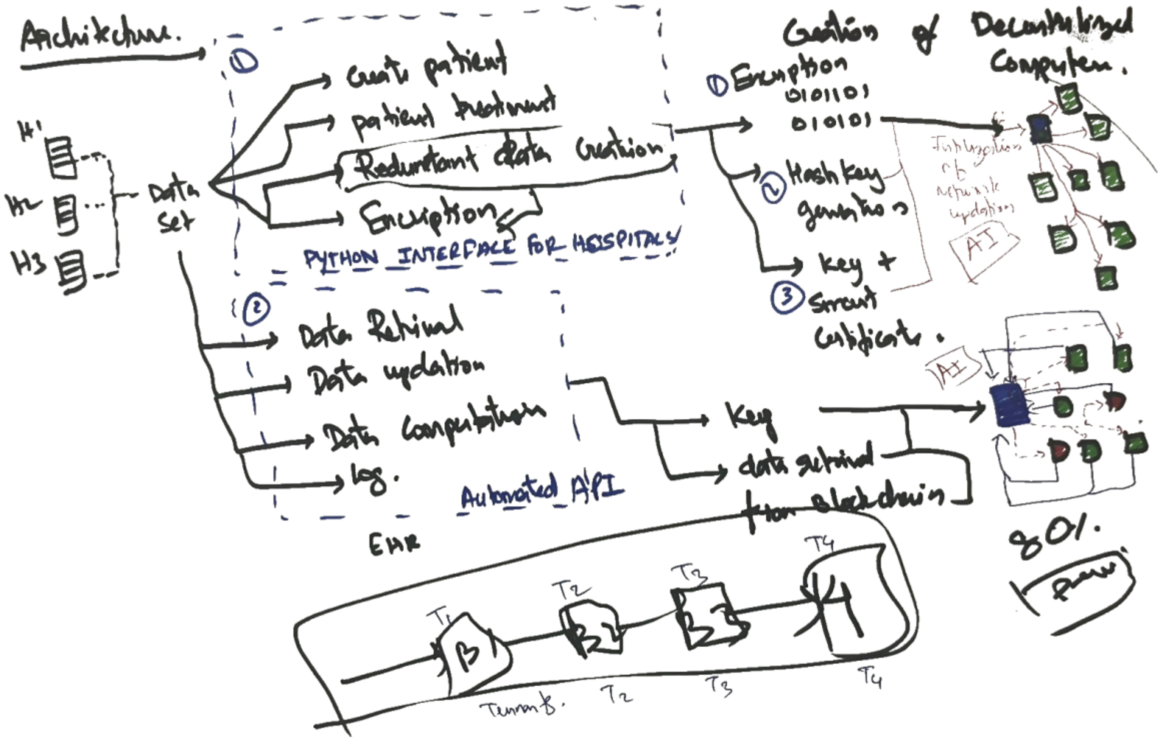
Overall Flow:

- Data is written to the RAID 1 sets for redundancy and mirrored storage.
- RAID 0 striping enhances performance by distributing data across multiple drives.
- The Monitoring AI oversees the entire system, ensuring optimal functionality.
- Log storage captures and encrypts relevant events for future reference and analysis.

Architecture

COIN TRANSACTION ARCHITECTURE





CONCLUSION

Results

Data Latency Calculation:

1. **Define the Starting Point (t_start) and Ending Point (t_end):**
   - **Starting Point (t_start):** The timestamp when data processing or transmission begins.
   - **Ending Point (t_end):** The timestamp when data processing or transmission is completed.
2. **Calculate Latency (L):**
   - **Latency (L):** The time taken for data to travel from the starting point to the ending point.
   - L=tend−tstartL=tend−tstart

Example:

Let's say you want to measure the latency for a data transfer operation:

- tstarttstart (Starting Point): 09:00:00 AM
- tendtend (Ending Point): 09:00:10 AM

Using the formula:

L=tend−tstartL=tend−tstart

L=09:00:10−09:00:00=10 secondsL=09:00:10−09:00:00=10 seconds

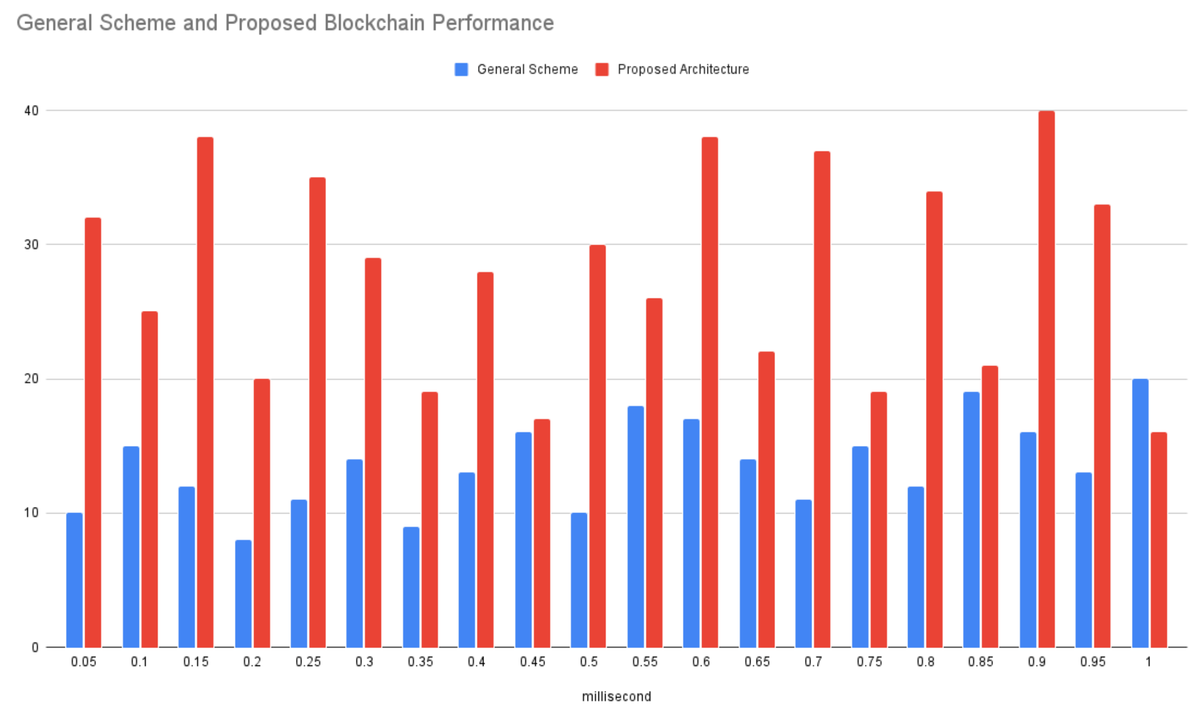
So, the data latency for this operation is 10 seconds.

Additional Considerations:

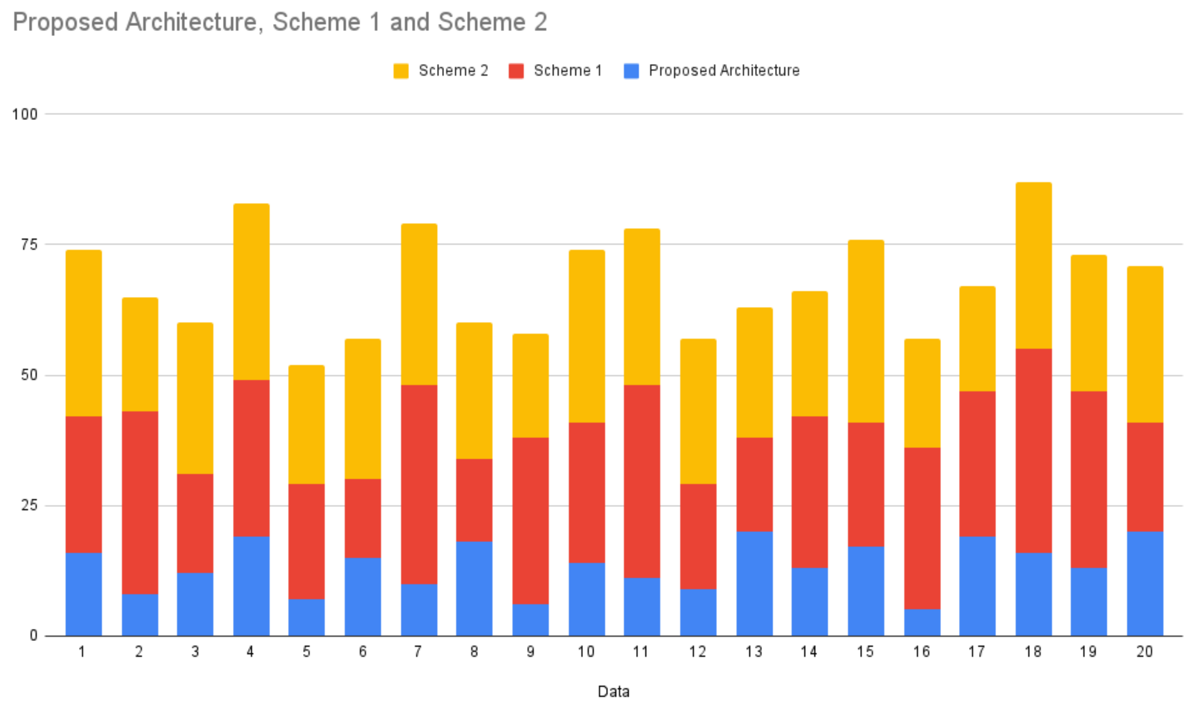
1. **Average Latency (L_avg):**
   - In scenarios where multiple data points are available, you may want to calculate the average latency.
   - Lavg=∑i=1nLinLavg=n∑i=1nLi, where LiLi is the latency for each operation and nn is the total number of operations.
2. **Latency Over Time:**
   - Analyzing latency over time can provide insights into system performance trends.
   - You may calculate latency at regular intervals and create a latency profile over time.

3. **Factors Affecting Latency:**
    - Understand the factors influencing latency, such as network congestion, processing time, and queuing delays.
    - Consider measuring round-trip latency for systems involving communication between two points.
4. **Use Appropriate Units:**
    - Ensure consistency in time units (seconds, milliseconds, microseconds) depending on the precision required for your analysis.

In a standard multitenant environment, data access and processing may be subject to latency, impacting the speed at which operations are executed. Our secured blockchain multitenancy leverages the decentralized and distributed nature of blockchain technology to significantly reduce latency. By distributing data across a secure and immutable ledger, the system minimizes the time it takes for data transactions to be validated and recorded.



The blue represents the time of delay retrival and the red represent the data from generic tennant. These are the time output of the runed and processed data where it proves that our proposed system take less time to process and retrieve data & results.



In the comparative analysis of three database architectures—Scheme 1 (Multiple Databases, Multiple Tenants Per Database, Shared Schema), Scheme 2 (Database Per Tenant), and our Proposed Architecture leveraging blockchain technology—the focus is on assessing the time required to decrypt data during a simulated brute force attack initiated from a Kali system. The resulting graph provides compelling insights into the security performance of each architecture.

Scheme 1 - Multiple Databases, Multiple Tenants Per Database, Shared Schema: In this scheme, multiple tenants share a database, each with its own set of data. The shared schema facilitates multi-tenancy but potentially exposes data to security threats. The graph illustrates the time taken for decryption attempts, revealing the vulnerability of this architecture to external attacks.

Scheme 2 - Database Per Tenant: In the second scheme, each tenant has a dedicated database, providing enhanced isolation. This approach aims to improve security by minimizing the impact of potential breaches. The graph reflects the decryption time under a brute force attack, providing insights into the resilience of the architecture.

Proposed Architecture - Blockchain-Based: Our proposed architecture leverages blockchain technology to secure data. The graph demonstrates the time taken for decryption attempts in this system, showcasing the enhanced security features. Blockchain's decentralized and tamper-resistant nature significantly contributes to a more resilient defense against external attacks.

Security Comparison and Conclusive Insights: The graphed data clearly indicates that our proposed architecture, utilizing blockchain technology, outperforms the traditional database architectures in terms of resistance to brute force attacks. The decentralized and immutable nature of the blockchain adds an additional layer of security, making unauthorized access significantly more challenging. As evidenced by the graph, the proposed architecture stands as a more secure solution, safeguarding sensitive data from external threats more effectively than the other two schemes.