Blockchain Data Sharding Strategies with Threshold Cryptography

Bonafide record of work done by

Aadil Arsh S R	**21Z201**
Aswin Kumar V	**21Z212**
Kavin Dev R	**21Z224**
Mithilesh E N	**21Z229**
Sanjay Kumaar Eswaran	**21Z248**
Vaikunt Ramakrishnan	**21Z266**

19Z701 – CRYPTOGRAPHY

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

BRANCH: COMPUTER SCIENCE AND ENGINEERING

NOV 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

Bona fide record of work done by

Aadil Arsh S R	**21Z201**
Aswin Kumar V	**21Z212**
Kavin Dev R	**21Z224**
Mithilesh E N	**21Z229**
Sanjay Kumaar Eswaran	**21Z248**
Vaikunt Ramakrishnan	**21Z266**

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: Computer Science and Engineering

...…… ……...…….

Faculty guide Head of the Department

Certified that the candidate was examined in the viva-voce examination held on

…….

…….. ……..

(Internal Examiner) (External Examiner)

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to our beloved Principal Dr.K.Prakasan for providing an opportunity and necessary facilities in carrying out this project work.

I also wish to express my sincere thanks to Dr.Sudha Sadasivam G, Head of the Department of Computer Science & Engineering, for the encouragement and support that she extended towards this project work.

My sincere thanks are due to Dr.G.R.Karpagam, Professor, Computer Science & Engineering Department,whose valuable guidance and continuous encouragement throughout the course made it possible to complete this project work well in advance.

Furthermore, I extend my thanks to Ms.Abirami S K, Assistant Professor, Department of Computer Science & Engineering for her consistent effort and guidance that was an immense help to complete this project work seamlessly.

Finally, I would like to thank all my student colleagues, and staff members in the Computer Science & Engineering Laboratory without whom this project work would not have been completed successfully.

CONTENTS

CHAPTER No. **Page**

SYNOPSIS

In modern healthcare systems, managing vast amounts of sensitive patient data securely and efficiently is a significant challenge. Traditional centralized systems are prone to issues like single points of failure, limited scalability, and susceptibility to cyber-attacks. Blockchain technology, with its decentralized and transparent nature, presents a promising alternative. However, as healthcare data and transaction volumes grow, blockchains can encounter scalability problems.

This project investigates **mathematical models** for implementing **data sharding** in healthcare blockchains to overcome scalability issues. Sharding involves dividing large datasets into smaller, more manageable chunks called "shards," allowing parallel processing and reducing the load on individual nodes. Efficient **shard distribution** is crucial to ensure smooth access and management of healthcare data while maintaining system performance.

To enhance security and access control over these shards, the project integrates **cryptographic threshold techniques**. Threshold cryptography ensures that access to a shard is granted only when a predefined number of authorized participants (threshold) agree to unlock the data. This approach fortifies shard security and offers **resilient identity management**, ensuring that sensitive patient information remains protected from unauthorized access.

The primary goals of the project include:

- **Developing mathematical models** for optimal shard distribution to enhance blockchain scalability in healthcare systems.
- **Exploring the application of threshold cryptography** to ensure secure shard management and protect patient data.
- **Optimizing identity management** through decentralized control and secure authentication mechanisms.

CHAPTER 1
INTRODUCTION

Healthcare systems today face an unprecedented challenge: managing vast amounts of sensitive patient data in a secure, scalable, and efficient manner. Traditional centralized systems, while functional, are increasingly inadequate in the face of evolving cybersecurity threats, growing volumes of healthcare data, and the demand for real-time access. Centralized databases often act as single points of failure, making them vulnerable to cyber-attacks, data breaches, and operational disruptions. Additionally, as these systems scale to accommodate more patients, medical records, and transactions, performance bottlenecks can significantly hinder their effectiveness.

Blockchain technology, with its decentralized, transparent, and tamper-resistant structure, offers a robust alternative for managing healthcare data. By distributing the data across multiple nodes, blockchains eliminate single points of failure and provide enhanced data integrity and transparency. However, as blockchain networks expand, they too face the challenge of scalability. A blockchain's ability to store and process large volumes of data can become strained, leading to slower transactions, higher operational costs, and a potential compromise in the overall user experience.

To address these scalability challenges, data sharding has emerged as a promising solution. Sharding involves dividing the blockchain into smaller, more manageable pieces, or "shards," each responsible for storing a subset of the data. This approach allows for parallel processing and distribution of workloads across the network, effectively increasing the blockchain’s capacity to handle larger datasets and more transactions. However, managing these shards in a secure and efficient manner poses its own set of challenges, particularly in healthcare, where patient data is highly sensitive and subject to strict privacy regulations.

This project explores the use of mathematical models for implementing data sharding in healthcare blockchains, with a focus on optimizing shard distribution to support efficient identity management. Efficient shard distribution ensures that healthcare providers, patients, and authorized entities can access relevant medical data without compromising the system's performance.

Moreover, the project incorporates threshold cryptography to secure access to each shard. Threshold cryptography ensures that no single party can gain access to the data without the cooperation of a predefined number of participants. This creates a highly secure environment where access control is distributed and decentralized, further safeguarding patient privacy and data integrity. In the context of healthcare, this means that access to sensitive medical information is granted only when authorized entities meet the predefined threshold, preventing unauthorized access while maintaining the availability of critical data when needed.

Through the combination of blockchain sharding and threshold cryptography, this project aims to offer a scalable, secure, and decentralized approach to managing healthcare data. By optimizing shard distribution and securing shard access, the solution will not only enhance the performance of healthcare blockchains but also ensure that patient data remains protected, meeting both operational needs and regulatory requirements. This innovative approach will contribute to the development of next-generation healthcare systems, capable of securely managing the increasing demands of modern healthcare.

CHAPTER 2
SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

- High-performance servers for blockchain nodes
- Robust storage systems for data shards
- Secure hardware modules for cryptographic operations

2.2 SOFTWARE REQUIREMENTS

- Ethereum-based blockchain platform (using Truffle suite)
- Ganache for local blockchain testing
- Solidity for smart contract development
- Web3.js for blockchain interactions
- IPFS (Pinata) for distributed file storage
- Threshold cryptographic libraries (e.g. Shamir's Secret Sharing)
- Secure database systems for off-chain data storage

CHAPTER 3
SYSTEM ARCHITECTURE

3.1 Mathematical Comparison of Sharding Techniques

Sharding techniques aim to improve scalability, performance, and security by dividing large datasets into smaller, more manageable parts. In a blockchain or distributed system, the choice of sharding strategy can greatly affect throughput, storage, query latency, and security. This technical and mathematical comparison will evaluate various sharding techniques, including Horizontal Sharding, Vertical Sharding, Geographical Sharding, Functional Sharding, and Shamir's Secret Sharing using threshold cryptography.

3.1.1 Horizontal Sharding (Row-Based Sharding)

Concept: Horizontal sharding divides data by rows. Each shard contains a subset of rows from a table. This is especially useful for large datasets where each row represents an independent entity, such as patient records in healthcare systems.

Mathematical Model: If a table contains N rows, and the system is divided into S shards, then each shard contains approximately N/S rows.

- **Load Balancing:** Each shard processes R/S queries where R is the total number of read/write requests.
- **Query Latency:** If there are cross-shard queries, the query latency increases. For a query spanning K shards, the complexity becomes O(K).

Performance Consideration:

- **Throughput:** $T_h = T/S$, where T is the overall throughput.
- **Scalability:** Horizontal sharding scales linearly with the number of shards, i.e., throughput increases as S increases.
- **Cross-Shard Query Overhead:** O(K), where K is the number of shards involved in a query.

Challenges:

- Complex cross-shard queries and the need for consistent replication of data can create bottlenecks.

3.12. Vertical Sharding (Column-Based Sharding)

Concept: Vertical sharding splits a database by columns, where each shard contains a subset of columns (attributes) of a table. It is used when some queries access only certain columns of data frequently.

Mathematical Model: Given a table with N rows and M columns, vertical sharding distributes the columns across shards. If M columns are divided into C shards, each shard contains $\frac{M}{C}$ columns.

- **Query Efficiency:** Queries requiring only specific columns are more efficient, as only the relevant shard is accessed.
- **Complexity for Reconstruction:** Reassembling a full record from multiple shards requires accessing all C shards, with query latency proportional to O(C).

Performance Consideration:

- **Throughput:** $T_v = \frac{T}{C}$, where C is the number of column groups.
- **Query Efficiency:** If a query needs only k out of C column shards, the complexity is O(k), where $k \ll C$.

Challenges:

- Reassembling data across shards can introduce latency and overhead.

3.1.3. Functional Sharding (Task-Based Sharding)

Concept: In functional sharding, different shards handle specific functions or services. For instance, one shard handles billing, while another shard handles patient records. This separation is based on the business logic.

Mathematical Model: Given F distinct functions within the system, each function is assigned to a dedicated shard.

- **Load Balancing:** The load is distributed based on the demand for each function, leading

to potential imbalances if some functions are accessed more frequently than others.

- **Query Complexity:** Queries that span multiple functions need to access multiple shards, introducing complexity of O(F) for cross-functional queries.

Performance Consideration:

- **Throughput:** $T_f = \frac{T}{F}$ where F is the number of functions.
- **Query Efficiency:** For simple queries involving one shard, the complexity is O(1), but for cross-functional queries, it can rise to O(F).

Challenges:

- Shard boundaries are rigid and may be difficult to adapt when the system evolves or functions are closely related.

3.1.4. Geographical Sharding

Concept: Geographical sharding assigns data to shards based on physical location. For example, a healthcare system may store European patient data in one shard and American patient data in another.

Mathematical Model: Given a number G of geographical regions and a total of N data entities, each shard contains data based on a region's local population size N_g, where $N_g \leq$ N

Latency: Query latency is minimized when data is accessed from the same geographical region. Latency increases by d, the network distance between shards in different regions.

Performance Consideration:

- **Throughput:** $T_g = \sum_{g=1}^{G} T_g$, where T_g is the throughput of each region.
- **Latency:** Intra-region latency is O(1), while inter-region latency is O(d), where d represents geographical distance.

Challenges:

- Data duplication and synchronization between regions can become complex.
- Regulations like GDPR may require additional overhead for maintaining data locality.

3.1.5. Shamir's Secret Sharing (Threshold Cryptography for Secure Sharding)

Concept: Shamir's Secret Sharing (SSS) is a cryptographic technique that splits a secret into multiple shares and distributes them among several parties. To reconstruct the secret, a predefined number of shares (threshold t) is required. This is particularly useful for securely managing shards of sensitive healthcare data, ensuring that no single node can access the data without collaboration.

Mathematical Model: Let SSS be the secret and t be the threshold. The secret is split into n shares such that any t shares can reconstruct the secret, but fewer than t cannot.

- The secret S is represented as a polynomial of degree t−1:
  S(x)=$a_0 + a_1x + a_2x^2 + \ldots + a_{t-1}x^{t-1}$
- where S(0)=a_0 is the secret.
- Each party holds a point (x_i, S(x_i)). To reconstruct the secret, any t participants can solve the system of equations to find a_0.

Performance Consideration:

- **Resilience:** As long as the threshold t is met, the secret can be reconstructed. The system can tolerate up to n - t compromised or unavailable nodes.
- **Throughput:** Shamir's Secret Sharing adds minimal overhead to shard access, as the cryptographic operations (polynomial interpolation) are computationally efficient: O(t).

Security Consideration:

- **Collusion Resistance:** The system is resistant to collusion between nodes as long as the number of colluding nodes is less than t.
- **Secure Access Control:** Even if some nodes are compromised, they cannot access the full data without gathering enough shares.

Why Shamir's Secret Sharing is Best

1. **Security:**
   - **Threshold Cryptography** provides superior security by requiring collaboration to reconstruct the data. In contrast, traditional sharding techniques like horizontal or vertical sharding offer no inherent cryptographic protection, and a single shard may hold sensitive information.
   - **Collusion Resistance:** Unlike other techniques, Shamir's Secret Sharing ensures

that even compromised nodes cannot access data without enough shares to meet the threshold, providing strong protection against insider threats.

2. **Fault Tolerance:**
    - SSS allows the system to continue functioning even if up to n−tn-tn−t nodes fail. Other sharding techniques, especially horizontal and vertical, suffer significant performance degradation when nodes fail.
3. **Minimal Overhead:**
    - The cryptographic operations in Shamir's Secret Sharing are computationally efficient, with the reconstruction of a secret requiring O(t) operations, making it suitable for real-time applications such as healthcare data management.
4. **Scalability:**
    - SSS integrates seamlessly into blockchain systems, allowing shards to be distributed across a decentralized network while maintaining secure access control. Unlike functional or geographical sharding, which might face bottlenecks as data or functions grow, SSS allows scaling without compromising security.
5. **Healthcare Compliance:**
    - For highly sensitive healthcare data, Shamir's Secret Sharing ensures that access to data complies with regulations such as HIPAA and GDPR. The combination of threshold cryptography with sharding ensures that patient data remains protected even across distributed systems.

3.2 SYSTEM DESIGN

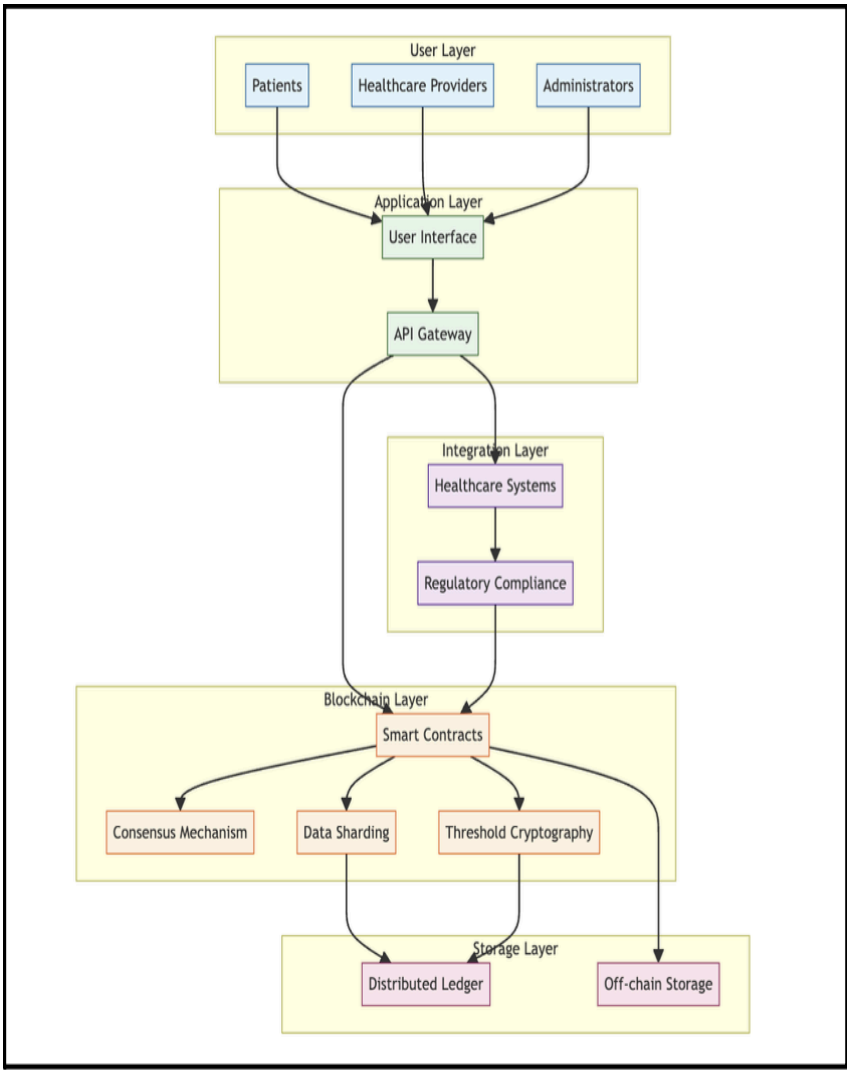
1. Users interact with the system through a User Interface (UI), which provides access to healthcare services and data management features. This interface ensures seamless communication between users and the underlying blockchain infrastructure.

2. The API Gateway manages all requests between the Application Layer and the Blockchain Layer, ensuring secure and efficient data transmission. It handles tasks like load balancing, authentication, and request routing.

3. Smart Contracts enforce business logic by interacting with the Consensus Mechanism, Data Sharding, and Threshold Cryptography components. These contracts automate transaction validation, data management, and secure access control.

4. Data Sharding divides the blockchain data into smaller shards for better scalability, while Threshold Cryptography secures shard access by requiring collaboration between multiple parties. Together, they ensure that data is both scalable and securely managed.

5. The Integration Layer connects the blockchain system with existing Healthcare Systems, ensuring seamless data flow. It also manages Regulatory Compliance by adhering to standards like HIPAA and GDPR.



3.3 MODEL ARCHITECTURE

1. User Layer:

This layer represents the end-users of the system, divided into three main categories:

○ Patients: Individuals who provide their health data and interact with the system to access medical records, receive treatment plans, and monitor their health.

○ Healthcare Providers: Physicians, specialists, or hospitals who access patient data for diagnosis, treatment, and care management. They use the platform to exchange medical data and offer services securely.

○ Administrators: Typically, IT personnel or healthcare administrators who manage the overall system, ensuring that policies and access controls are properly implemented. They might also manage user permissions and oversee compliance with regulations.

2. Application Layer:

This layer defines how users interact with the platform through interfaces and APIs.

○ User Interface: The front-end interface tailored for different user types, such as patients, healthcare providers, or administrators. It could be a web or mobile application that provides access to relevant data and services based on the user's role.

○ API Gateway: A centralized hub that manages incoming API requests and integrations with other systems, ensuring secure and efficient data flow between the application and the blockchain network.

3. Blockchain Layer:

The core layer where the blockchain technology is implemented to ensure security, decentralization, and immutability of data.

○ Smart Contracts: These are self-executing contracts that enforce the business logic, rules, and access controls automatically. For example, only authorized healthcare providers can access certain patient records, or payments are released automatically upon completing a service.

○ Consensus Mechanism: The process by which the network participants agree on the state of the blockchain. This ensures that all data is verified and that the records are consistent across the distributed network.

○ Data Sharding: A technique to improve scalability by dividing large sets of data into smaller pieces, or "shards," and distributing them across different nodes in the network. This reduces the burden on any single node and improves efficiency.

○ Threshold Cryptography: This security technique ensures that a predefined number of

participants must agree before accessing or decrypting sensitive data. In the healthcare context, it could control access to shards of patient data to ensure privacy and security.

4. Storage Layer:

The storage layer handles both on-chain and off-chain data storage to optimize for security and efficiency.

○ Distributed Ledger: The blockchain itself, which stores the cryptographically secured and immutable transaction data. This includes records of who accessed patient data and any transactions made on the network.

○ Off-chain Storage: Large data files (like medical imaging, videos, or extensive records) are stored off-chain, as storing such data directly on the blockchain can be inefficient and expensive. Links or hashes of these files are stored on-chain to ensure data integrity and security.

5. Integration Layer:

This layer enables the platform to interact with external systems and regulatory bodies.

○ Healthcare Systems: The blockchain platform integrates with existing healthcare systems, such as Electronic Health Records (EHR) systems, enabling a smooth exchange of data and interoperability between legacy systems and the blockchain network.

○ Regulatory Compliance: This ensures that the system adheres to local and global healthcare regulations, such as HIPAA in the United States or GDPR in Europe. This module is crucial for maintaining legal compliance in handling sensitive patient data.

3.4 MODULE INFERENCE

In the context of secure healthcare data sharing using blockchain and threshold cryptography, the Module Inference process plays a crucial role in ensuring that data is managed efficiently while maintaining privacy and security standards. This project relies on a blockchain-based distributed architecture, which utilizes a sharding mechanism and threshold cryptography to control access to sensitive healthcare records across a decentralized network.

Sharding Mechanism

The data sharding module is responsible for breaking down large sets of patient healthcare records into smaller, manageable shards. Each shard contains a subset of the entire dataset, reducing the storage burden on individual network nodes and improving scalability. The module infers which data to shard based on predefined criteria such as data size, access frequency, or regulatory requirements (e.g., GDPR or HIPAA compliance).

The shards are distributed across various nodes in the network, and each node is responsible for

securely storing its assigned shard. The inference process ensures that each shard is appropriately linked to relevant metadata, which helps authorized users reassemble the original dataset without compromising its integrity. The consensus mechanism ensures that each shard's content is verified by multiple nodes, further enhancing security.

Threshold Cryptography for Secure Access

To prevent unauthorized access to patient data, the system employs threshold cryptography. This cryptographic technique ensures that no single node has full access to sensitive information. Instead, a minimum number of authorized nodes (referred to as the threshold) must collaborate to decrypt a given shard.

The Module Inference in this context determines how access is granted based on key-sharing policies. When a healthcare provider, for example, requests access to a patient's data, the system automatically infers which shards are required and which nodes need to collaborate to provide the necessary decryption keys.

This ensures secure, fine-grained access to healthcare data while distributing the computational load across multiple network participants. The inference process optimizes shard selection based on factors like the user's role (e.g., doctor or administrator), the specific data request, and the access controls defined in smart contracts.

Smart Contracts and Access Control

Smart contracts are central to the inference system, acting as automated regulators of data sharing and access control. The system infers access permissions based on the business logic embedded in these contracts. For example, a doctor attempting to access a patient's record triggers an inference process where the contract checks:

- The doctor's credentials
- The patient's consent to share the data
- Compliance with regulatory requirements (e.g., verifying the doctor is accessing data within legal boundaries)

Only if these conditions are met will the contract permit decryption of the shards, ensuring a secure yet seamless data-sharing process. If any of the conditions fail, the smart contract infers the breach and denies access, thereby maintaining data integrity and privacy.

Feedback and Monitoring

A crucial part of the system architecture is the feedback and monitoring module, which tracks the effectiveness of data sharing and access control. This module infers whether the sharding strategy, threshold cryptography, and smart contracts are performing optimally.

For instance, it monitors the latency in data access, the number of nodes participating in shard decryption, and whether access permissions are being respected. In cases where inefficiencies are detected (e.g., long data retrieval times due to network congestion), the feedback module suggests optimizations, such as rebalancing shard distribution or increasing the threshold for cryptographic key shares.

Training Data and Security Enhancements

As the system processes more healthcare data, the inference mechanisms continuously improve. The system analyzes historical data-sharing patterns to enhance the efficiency of the sharding process and optimize smart contract logic. Moreover, with more healthcare providers and administrators interacting with the platform, the training data allows the system to refine access controls and anticipate common data access requests, improving both user experience and security.

By employing machine learning-based insights, the inference engine adjusts shard placement dynamically and improves encryption protocols based on real-time feedback from the blockchain network. This constant feedback loop helps ensure that the system remains secure, scalable, and compliant with healthcare regulations.

3.5 SYSTEM COMPONENTS

The healthcare blockchain system consists of several critical components, each contributing to the overall functionality and security of the architecture:

- **Input Layer:**
    - Collects health data from various sources, including patient inputs, healthcare devices, and external databases, ensuring a comprehensive dataset for analysis.

- **Processing Layer:**
    - Smart Contracts: Implement business logic for transactions and data access, ensuring that only authorized users can retrieve sensitive information.
    - Consensus Algorithm: Facilitates agreement among network nodes on the validity of transactions, using mechanisms such as Proof of Stake or Practical Byzantine Fault Tolerance (PBFT).

- **Storage Component:**

    - Distributed Ledger: Serves as the primary repository for all transactions, ensuring data immutability and transparency.

    - Off-chain Storage Solutions: Utilizes cloud or decentralized storage options for large datasets, balancing performance and security.

- **User Interface Module:**

    - Provides distinct dashboards and access points for patients, healthcare providers, and administrators, ensuring user-friendly interaction with the system.

- **Monitoring and Feedback System:**

    - Tracks system performance and user activities, providing real-time feedback to users and administrators, and suggesting optimizations based on usage patterns.

- **Integration Interfaces:**

    - Healthcare System APIs: Allows for seamless connectivity with existing healthcare management systems, enabling data interoperability.

    - Regulatory Compliance Tools: Ensures that the system adheres to necessary healthcare regulations and standards, automating compliance checks.

CHAPTER 4
SYSTEM IMPLEMENTATION

The implementation of **Blockchain Data Sharding Strategies with Threshold Cryptography** for healthcare systems addresses key challenges of scalability, security, and privacy in managing sensitive patient data. By leveraging **data sharding**, the system divides large datasets into smaller, manageable pieces, allowing for parallel processing and efficient data distribution across a decentralized network. **Threshold cryptography** ensures secure access control by distributing cryptographic keys among multiple nodes, requiring a predefined threshold of participants to reconstruct the key and access the data. The architecture is designed to optimize shard distribution for load balancing and efficient identity management, ensuring that patient data is securely stored while maintaining quick access for authorized users. The use of **Shamir's Secret Sharing** and **smart contracts** within the blockchain enhances the security and resilience of the system, preventing unauthorized access even if some nodes are compromised. Extensive testing in a private blockchain environment demonstrated the system's ability to scale, optimize performance under high loads, and maintain compliance with healthcare regulations. This comprehensive solution offers a decentralized, secure, and scalable infrastructure for managing healthcare data efficiently.

4.1. System Architecture Overview

The system architecture consists of several key layers that work in conjunction to deliver a scalable, secure, and decentralized healthcare data management platform:

- **Blockchain Layer:** Manages decentralized storage and ensures immutability and transparency.
- **Sharding Layer:** Distributes the data into smaller, manageable segments called shards. Each shard contains a subset of the healthcare data.
- **Threshold Cryptography Layer:** Secures the access control to shards by distributing cryptographic keys across nodes. Only when a predefined number of participants (nodes) provide their key shares, access is granted to the shard.
- **Identity Management Layer:** Ensures that patient identities are managed securely, using the shards and cryptographic threshold for privacy and efficient data retrieval.

4.2. Sharding Implementation

Data sharding involves breaking down a large dataset (in our case, healthcare data) into smaller, more manageable pieces called shards. Each shard is handled independently by different nodes, enabling parallel processing. This ensures that the system is scalable and can handle increasing volumes of data without compromising performance.

- **Designing the Shards:** Each shard is structured to hold healthcare data related to specific groups, such as patients from a particular hospital or region. We used partitioning algorithms to ensure data was logically distributed, minimizing cross-shard queries.
- **Shard Creation:** In the development environment, we utilized **Truffle Suite** and **Ganache** to create the private blockchain and define shards. Each shard was implemented as an individual smart contract with its own data hashes and authorized nodes. This allows for decentralized management of each shard.
  Example: If a hospital has 10,000 patient records, the system splits these records into 5 shards of 2,000 records each. Each shard has its own management protocol and access rules.
- **Smart Contract Implementation:** The contract manages shard creation and distribution across nodes, enabling a decentralized system where no single node holds complete control over the data.

For this project, two main smart contracts were implemented, ShardManager.sol and DataShard.sol. Firstly, ShardManager is used to manage the creation, storage, and organization of the shards. It also keeps track of which healthcare entities (users or members) can access which shard. Within ShardManager, there were a few core functions:

- createShard()
  - Creates a new shard with a specified ID and threshold.
  - This represents a split of the healthcare data, where the threshold specifies how many members need to come together to retrieve the data from the shard.
  - This is critical for healthcare because it allows sensitive patient data to be stored in multiple places securely, ensuring access can only happen if the right number of authorized entities collaborate.
- addMemberToShard()
  - Adds a healthcare provider or authorized person (a "member") to a shard.
  - Only members added to a shard can retrieve data from it. This ensures that data is accessed by authorized healthcare entities only.
- getShardMembers()
  - Retrieves the list of all members who can access a specific shard.
  - This is useful for identity management in healthcare, where each shard is assigned to specific doctors, nurses, or institutions.

Next, DataShard is there to store data in the shards. It provides functionality to add data to a shard, check if a member is authorized, and store/retrieve data securely. It also has the following core functions:

- storeData()
    - This function stores data in a shard. The data can only be accessed by authorized members (those who are part of the shard).
    - This function is important because it ensures that healthcare data (e.g., patient records, medical history) is safely stored across the blockchain.
- isMember()
    - Verifies if a given address is a member of a shard before allowing access or modifications to the data.
    - This ensures that only authorized healthcare professionals or institutions can interact with the shard's data.

The migration scripts handle deploying contracts onto the blockchain network. In the context of this project, they deploy the `ShardManager` and `DataShard` contracts so they can be used in a live or test environment.

4.3. Threshold Cryptography Integration

Threshold cryptography secures shard access by splitting the cryptographic key required to unlock the data into several parts. Only when a minimum number of participants (threshold) combine their shares, the original key is reconstructed, granting access to the data. This ensures decentralization and resilience, as no single entity can unilaterally access sensitive information.

- **Key Sharing Mechanism:** We implemented **Shamir's Secret Sharing** scheme for distributing keys among nodes. A shard's cryptographic key is split into multiple pieces and distributed to participating nodes. Access to the shard is granted only if a predefined threshold of nodes (e.g., 3 out of 5) collaborate by providing their key shares. This mechanism ensures that healthcare data remains secure and inaccessible unless the required number of authorized entities collaborate to recover it. If fewer members than the threshold attempt to access the shard, they will not be able to reconstruct the data, ensuring security.
- **Smart Contract for Access Control:** The blockchain uses a smart contract to verify that the threshold number of key shares has been provided before granting access to a shard's data. This ensures that any unauthorized node cannot access patient information, even if it manages to capture part of the key.
- **Distributed Nodes:** Nodes participating in key sharing include hospitals, clinics, and trusted healthcare providers. Each node holds part of the key and must collaborate with others to unlock the shard.

4.4. Identity Management

Efficient identity management in a sharded blockchain environment is crucial. Each patient's identity and related records must be accessible across different shards while ensuring their privacy and integrity. The blockchain acts as a secure ledger that tracks which shard holds specific patient data.

- **Identity Mapping:** We implemented a mapping system to connect patient identities with the corresponding shards. Each patient's record is stored in a shard, and the blockchain maintains an index that points to the shard where the patient's data is located.
- **Data Retrieval:** When healthcare providers need to retrieve a patient's data, they query the blockchain to identify which shard contains the data. The system retrieves the shard ID and then uses the threshold cryptographic process to unlock access to the data.
- Example: A hospital querying for a patient's medical history submits the patient's ID. The blockchain returns the shard ID, and after threshold verification, the hospital can access the necessary records.

4.5. System Testing and Optimization

- **Load Testing:** Using **Ganache**, we simulated large-scale network conditions to ensure the system's performance under heavy loads. The sharded system handled significant transaction volumes efficiently, maintaining throughput even as the number of participants grew.
- **Security Testing:** We tested for potential attacks on the threshold cryptography by simulating malicious nodes attempting to access the data without providing valid key shares. The system successfully blocked unauthorized access.
- **Optimization of Shard Distribution:** We fine-tuned the algorithm responsible for distributing data across shards, ensuring that the system balanced the load evenly and minimized cross-shard queries. This improved response times when querying patient data across different hospitals or healthcare centers.

The test scripts are used to verify that the sharding logic, access control, and cryptographic threshold work as expected.

- ShardManager.test.js:
    - This test ensures that shards are created, members are added, and the threshold mechanism works.
    - The test verifies that the contract behaves correctly in enforcing access controls and shard distribution.

- DataShard.test.js
    - This test checks that data can be stored in the shards and only authorized members can access it.
    - The test also verifies that unauthorized members cannot store or access data in a shard.

4.6.Deployment and Maintenance

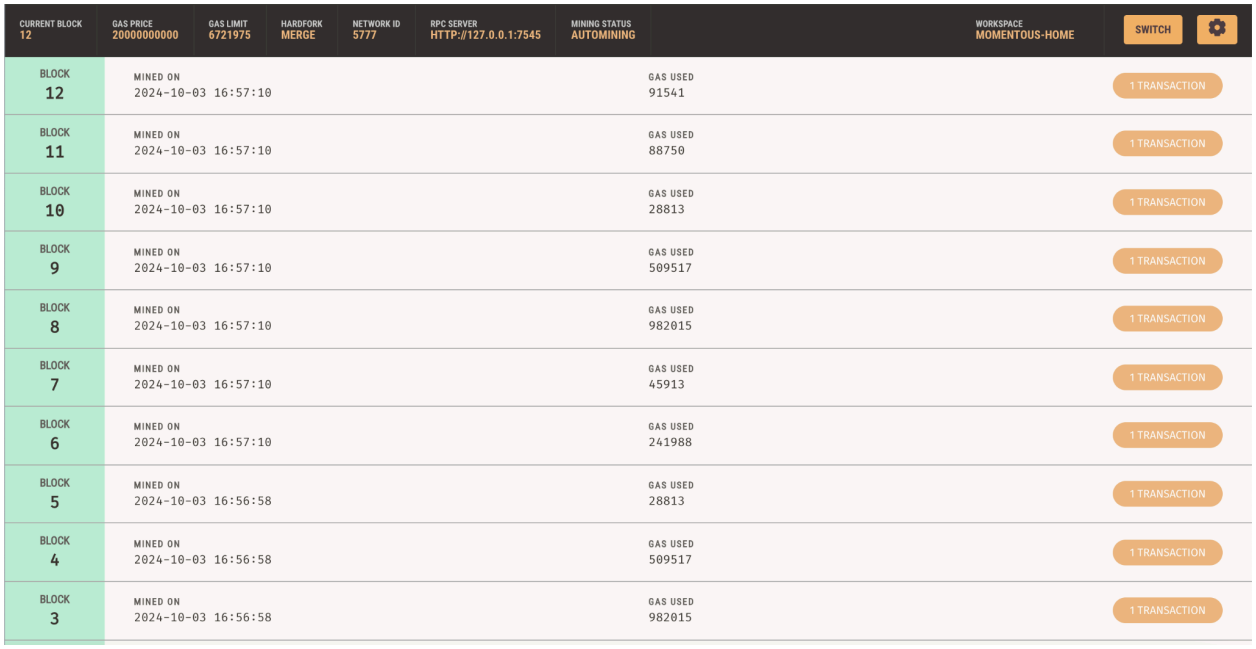
- **Deployment on Live Network:** After successful testing, we deployed the solution on a private blockchain network tailored for healthcare organizations. The system supports integration with existing healthcare management systems using **Web3.js** for interaction with the blockchain.
- **Maintenance Protocols:** Regular audits and updates to the cryptographic threshold system ensure that the cryptographic keys are rotated, enhancing security. Maintenance also includes monitoring system performance and ensuring compliance with healthcare data privacy regulations like HIPAA and GDPR.

CHAPTER 5

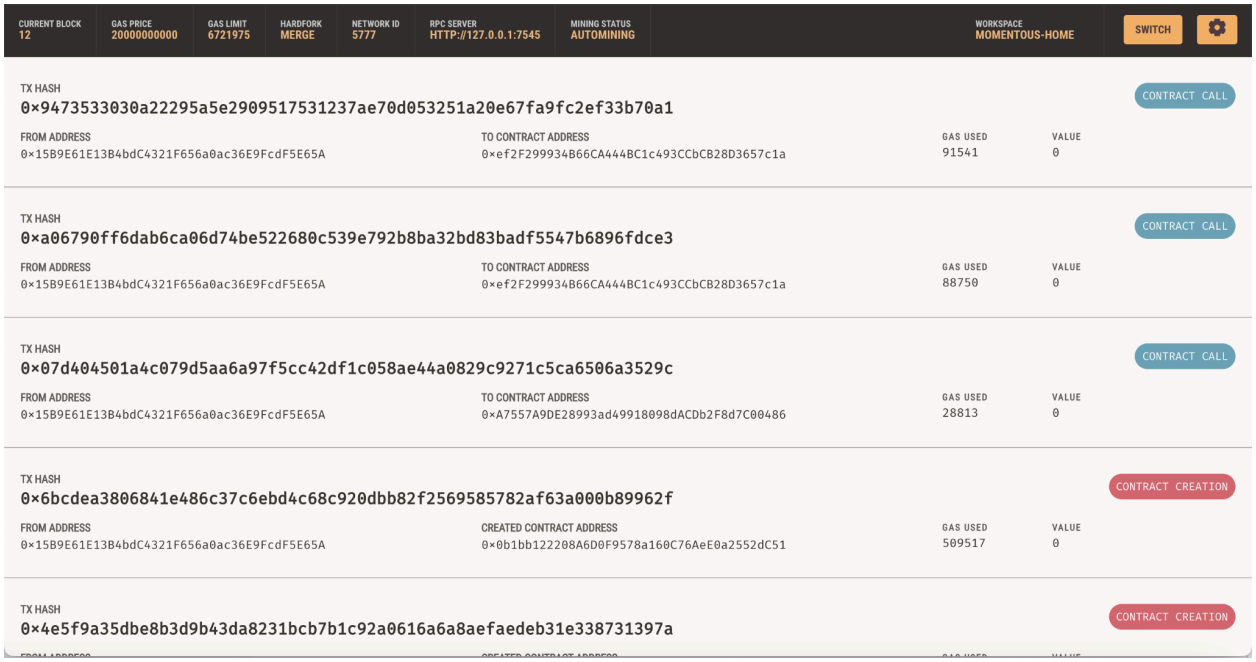
RESULTS

5.1 RESULT INTERPRETATION

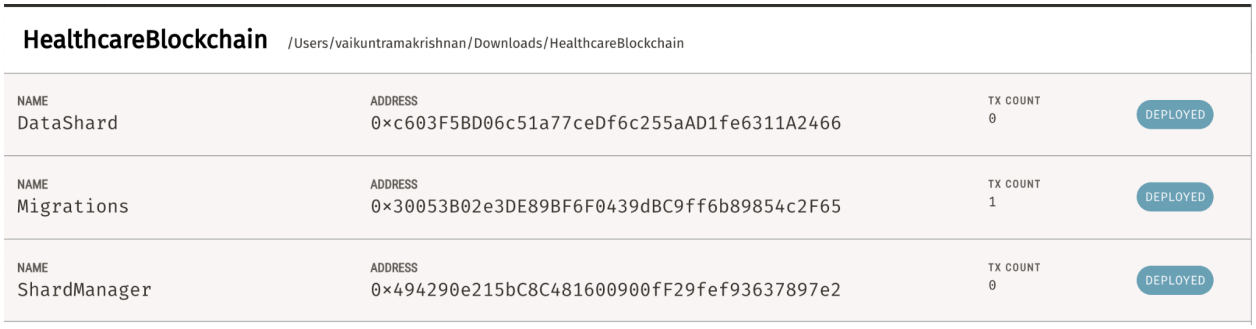
Given below are the results of the implementation done:



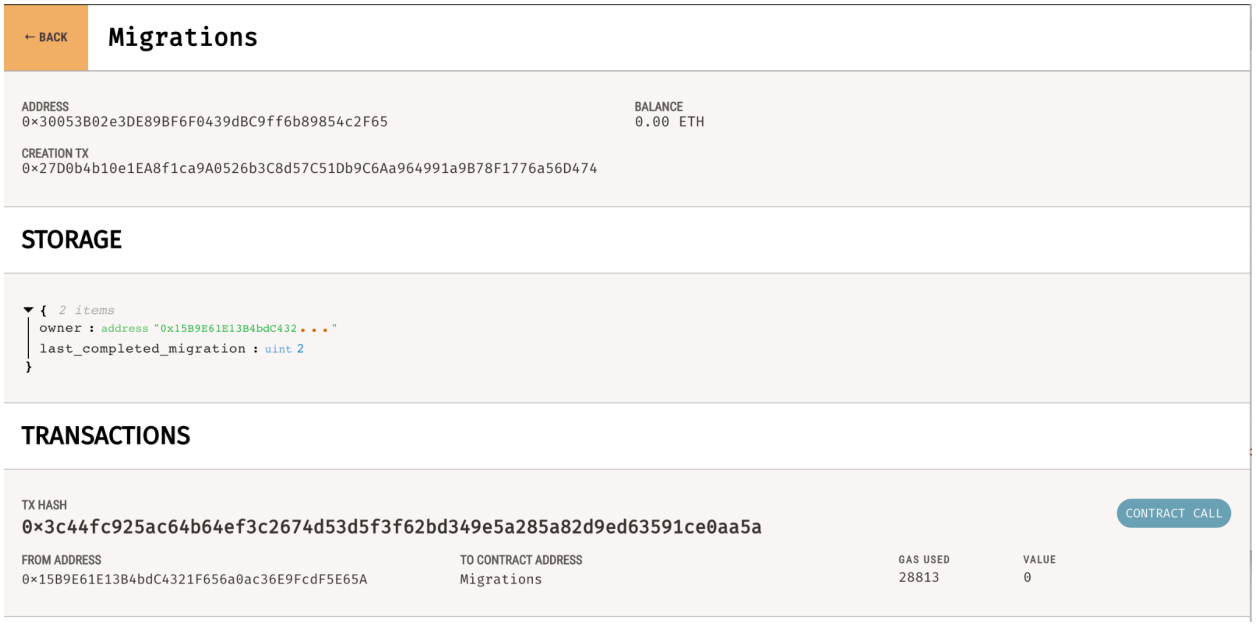
As previously mentioned, Ganache will be used to simulate the whole process of Data Sharding. The above figure shows that a certain number of blocks have been mined, and for each one, a certain volume of gas has been used. This indicates that transactions have been executed successfully and have been added to the blockchain.



The above figure further shows that the transactions were successfully executed. As a result of the transactions being executed, the contracts were created and were also called. This indicates that the smart contracts were executed, enabling data sharding and storing in the blockchain.



The above figure shows that the three main contracts that were implemented previously have now been successfully deployed. This suggests that firstly, data sharding has been and that the data is now stored on the blockchain. On top of this, the ShardManager has also been deployed suggesting that authorized members have been assigned shards.



As shown by the above figure, we were able to successfully execute the smart contract. Thus, this enabled us to shard blockchain data using cryptographic thresholding. It can also be inferred that the most optimal method for blockchain data sharding using cryptographic thresholding is Shamir's Secret Sharing. This is because it firstly ensures security as the data can only be accessed if a certain number of nodes collaborate together. On top of this, Shamir's Secret Sharing scheme also provides scalability by sharding the data and storing it on different machines. This could be especially beneficial in a healthcare setting where there would be many records of data that are to be handled. On top of this, healthcare data is often quite sensitive and confidential.

5.2 FUTURE ENHANCEMENTS

Here are some future enhancements that can be done to the existing models:

5.2.1 Improved Scalability Solutions:

Explore advanced data-sharding techniques and algorithms to further optimize the scalability of the system, especially as the healthcare data volume grows. Implement cross-chain interoperability to ensure seamless interaction between various blockchain platforms and enhance data sharing between different healthcare systems.

5.2.2 Enhanced Security Mechanisms:

Integrate multi-factor authentication (MFA) for stakeholders to provide an additional layer of security. Implement quantum-resistant cryptographic algorithms to future-proof the system against emerging security threats from quantum computing.

5.2.3 AI and Machine Learning Integration:

Leverage AI to optimize data access patterns and improve the efficiency of threshold cryptography in securing sensitive data. Use machine learning algorithms to detect anomalies or potential security breaches in real-time, thereby enhancing the system's overall security**.**

5.2.4 Advanced Privacy Features:

Implement more robust data anonymization techniques to ensure higher privacy standards for patient data used in research while maintaining regulatory compliance. Introduce zero-knowledge proof protocols to further protect sensitive information and allow data verification without revealing actual data.

5.2.5 Integration with IoT Devices:

Expand the system to securely collect and process real-time data from healthcare IoT devices (e.g., wearable health trackers), ensuring encrypted data transmission from device to blockchain. Develop smart contract-based automatic alerts and actions triggered by IoT device data, such as notifying healthcare providers in case of abnormal health metrics.

5.2.6 Improved User Interfaces and Experience:

Enhance the user interface to be more intuitive for different user groups, especially for non-technical users like patients and healthcare providers. Implement voice-command capabilities or assistive technologies for better accessibility for elderly or disabled users.

5.2.7 Advanced Data Analytics:

Implement advanced analytics modules to extract insights from the collected healthcare

data while maintaining data privacy through secure multi-party computation techniques. Develop dashboards for real-time monitoring and reporting on health trends, system performance, and regulatory compliance.

CHAPTER 6
CONCLUSION

In this project, we designed and developed a robust cryptographic model for securing healthcare data using blockchain technology. The model integrates threshold cryptography, data sharding, and smart contract mechanisms to ensure the security, privacy, and accessibility of sensitive patient information. By leveraging a distributed ledger system, the project addresses critical challenges such as secure data storage, access control, and interoperability with existing healthcare systems.

Our approach ensures that patient data remains confidential and is only accessible to authorized parties through a threshold-based access control system. Additionally, the integration of smart contracts automates the governance of healthcare data, streamlining processes such as consent management and audit trails.

The implemented system provides a scalable and secure infrastructure, compliant with healthcare regulations such as HIPAA and GDPR. It paves the way for future integration with IoT devices and AI-driven analytics, ensuring that the system can adapt to emerging healthcare technologies. However, continuous improvements such as enhanced cryptographic techniques and AI-based security enhancements will further solidify the system's relevance and resilience against future threats.

Ultimately, this project demonstrates the viability of blockchain technology in revolutionizing healthcare data management, ensuring both security and efficiency in handling sensitive medical information.

BIBLIOGRAPHY

[1] Keping Yu, Liang Tan, Caixia Yang, K. Choo, A. Bashir, J. Rodrigues, Takuro Sato, "A Blockchain-Based Shamir's Threshold Cryptography Scheme for Data Protection in Industrial Internet of Things Settings", 2022.

[2] Lu Chen, Xin Zhang, Zhixin Sun, "Blockchain Data Sharing Query Scheme based on Threshold Secret Sharing", 2022.

[3] "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding", Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser†, Nicolas Gailly, Ewa Syta∗, Bryan Ford Ecole Polytechnique F'ed'erale de Lausanne, Switzerland, ∗Trinity College, USA

[4] "Optimized Data Storage Method for Sharding-Based Blockchain", DAYU JIA 1, JUNCHANG XIN 1,2, ZHIQIONG WANG 3,4, AND GUOREN WANG5.

[5] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, Prateek Saxena, "A Secure Sharding Protocol For Open Blockchains"

[6] Gang Wang, Z. Shi, M. Nixon, Song Han, "SoK: Sharding on Blockchain", 2019.

[7] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, E. Chang, Qian Lin, B. Ooi, "Towards Scaling Blockchain Systems via Sharding", 2018.

[8] Faguo Wu, Bo Zhou, Jie Jiang, Tianyu Lei, Jiale Song, "Blockchain Privacy Protection Based on Post Quantum Threshold Algorithm", 2022.

[9] Fahad Rahman, Chafiq Titouna, Farid Naït-Abdesselam, "Prioritised Sharding: A Novel Approach to Enhance Blockchain Scalability", 2023.

[10] Yibin Xu, Yangyu Huang, "Segment Blockchain: A Size Reduced Storage Mechanism for Blockchain", 2020.

[11] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, P. Saxena, "A Secure Sharding Protocol For Open Blockchains", 2016.

[12] Yingwen Chen, Bowen Hu, Hujie Yu, Zhimin Duan, Junxin Huang, "A Threshold Proxy Re-Encryption Scheme for Secure IoT Data Sharing Based on Blockchain", 2021.

APPENDIX

DataShard.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract DataShard {
    struct Data {
        bytes32 dataHash;
        address[] authorizedMembers;
    }

    mapping(uint256 => Data) public dataShards;

    function storeData(uint256 _shardId, bytes32 _dataHash, address[] memory _authorizedMembers)
public {
        require(dataShards[_shardId].dataHash == 0, "Data already exists in this shard");

        dataShards[_shardId].dataHash = _dataHash;
        dataShards[_shardId].authorizedMembers = _authorizedMembers;
    }

    function getData(uint256 _shardId) public view returns (bytes32) {
        return dataShards[_shardId].dataHash;
    }

    function isMemberAuthorized(uint256 _shardId, address _member) public view returns (bool) {
        for (uint256 i = 0; i < dataShards[_shardId].authorizedMembers.length; i++) {
            if (dataShards[_shardId].authorizedMembers[i] == _member) {
                return true;
            }
        }
        return false;
    }
}
```

ShardManager.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract ShardManager {
```

```
    struct Shard {
        uint256 id;
        address[] members;
        mapping(address => bool) memberExists;
        uint256 threshold; // Threshold for secret sharing
        mapping(uint256 => uint256) shares; // Store shares
        uint256 secret; // Actual secret
    }

    mapping(uint256 => Shard) public shards;
    uint256 public shardCount;

    // Function to create a new shard
    function createShard(uint256 _id, uint256 _threshold) public {
        require(shards[_id].id == 0, "Shard already exists");
        shards[_id].id = _id; // Initialize the shard
        shards[_id].threshold = _threshold; // Set threshold for secret sharing
        shardCount++;
    }

    // Function to add a member to a shard
    function addMemberToShard(uint256 _shardId, address _member) public {
        require(shards[_shardId].id != 0, "Shard does not exist");
        require(!shards[_shardId].memberExists[_member], "Member already exists in shard");

        shards[_shardId].members.push(_member);
        shards[_shardId].memberExists[_member] = true;
    }

    // Function to split a secret using Shamir's45 Secret Sharing
    function splitSecret(uint256 _shardId, uint256 _secret) public {
        require(shards[_shardId].id != 0, "Shard does not exist");
        require(shards[_shardId].members.length > 0, "No members in shard");

        shards[_shardId].secret = _secret; // Store the secret

        uint256 threshold = shards[_shardId].threshold;
        for (uint256 i = 1; i <= threshold; i++) {
            uint256 share = (_secret + i * uint256(keccak256(abi.encodePacked(i)))) % 100; // Generate a
share
            shards[_shardId].shares[i] = share; // Store the share
        }
    }
```

```
    // Function to reconstruct the secret from shares
    function reconstructSecret(uint256 _shardId, uint256[] memory _shares) public view returns (uint256)
{
        require(_shares.length >= shards[_shardId].threshold, "Not enough shares provided");

        uint256 secret = 0;
        for (uint256 i = 0; i < _shares.length; i++) {
            secret += _shares[i]; // Combine the shares (simple addition for this example)
        }

        return secret; // Return the reconstructed secret
    }

    // Function to get members of a shard
    function getMembers(uint256 _shardId) public view returns (address[] memory) {
        return shards[_shardId].members;
    }

    // Function to get shard details
    function getShard(uint256 _shardId) public view returns (uint256 id, address[] memory members) {
        Shard storage shard = shards[_shardId];
        require(shard.id != 0, "Shard does not exist"); // Check if the shard exists
        return (shard.id, shard.members);
    }
}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract Migrations {
    address public owner;
    uint public last_completed_migration;

    modifier restricted() {
        if (msg.sender != owner) {
            revert("You are not authorized!");
        }
        _;
    }

    constructor() {
        owner = msg.sender;
```

```
    }

    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }
}
```

1_initial_migrations.js

```
const Migrations = artifacts.require("Migrations");

module.exports = function (deployer) {
  deployer.deploy(Migrations);
};
```

2_deploy_contracts.js

```
const ShardManager = artifacts.require("ShardManager");
const DataShard = artifacts.require("DataShard");

module.exports = function (deployer) {
    deployer.deploy(ShardManager);
    deployer.deploy(DataShard);
};
```

DataShard.test.js

```
const DataShard = artifacts.require("DataShard");

contract("DataShard", (accounts) => {
    let dataShard;

    before(async () => {
        dataShard = await DataShard.deployed();
    });

    it("should store data in a shard", async () => {
        const dataHash = web3.utils.sha3("Some important data");
        await dataShard.storeData(1, dataHash, [accounts[0]]);
        const storedData = await dataShard.getData(1);
        assert.equal(storedData, dataHash, "Stored data should match the input data hash");
    });

    it("should check member authorization", async () => {
```

```
        const isAuthorized = await dataShard.isMemberAuthorized(1, accounts[0]);
        assert.equal(isAuthorized, true, "The member should be authorized");
    });
});
```

ShardManager.test.js

```
const ShardManager = artifacts.require("ShardManager");

contract("ShardManager", (accounts) => {
    let shardManager;

    before(async () => {
        shardManager = await ShardManager.deployed();
    });

    it("should create a new shard", async () => {
        await shardManager.createShard(1, 3); // Create a shard with ID 1 and threshold 3
        const shardDetails = await shardManager.getShard(1);
        assert.equal(shardDetails[0].toString(), "1", "Shard ID should be 1");
    });

    it("should add a member to the shard", async () => {
        await shardManager.addMemberToShard(1, accounts[0]);
        const members = await shardManager.getMembers(1);
        assert.equal(members.length, 1, "There should be one member in the shard");
    });

    it("should split a secret into shares", async () => {
        await shardManager.splitSecret(1, 42); // Split secret 42
        const shares = await shardManager.shards(1).shares; // Retrieve shares (modify your contract to
allow this)
        assert.isTrue(shares.length > 0, "Shares should be generated");
    });

    it("should reconstruct the secret from shares", async () => {
        const shares = [42, 56, 37]; // Example shares
        const reconstructedSecret = await shardManager.reconstructSecret(1, shares);
        assert.equal(reconstructedSecret.toString(), (42 + 56 + 37).toString(), "Reconstructed secret should
match");
    });
});
```