SECURE HEALTHCARE DATABASE-AS-A-SERVICE (SECHEALTHDB)

Mounika Seelam
Kent State University
Department of Computer Science
Master of Science
mseelam2@kent.edu

Amitha Ajithkumar
Kent State University
Department of Computer Science
Bachelor of Arts
aajithku@kent.edu

Prajwal Devaraj
Kent State University
Department of Computer Science
Master of Science
pdevaraj@kent.edu

Abstract - The SecHealthDB system is designed to provide a secure and privacy-preserving healthcare Database-as-a-Service (DBaaS) model within a semi-trusted cloud environment. Because cloud database providers may execute operations correctly but still attempt to observe or analyze user data, strong confidentiality and integrity protections are essential. This project addresses these challenges by integrating AES-GCM encryption for sensitive attributes, bcrypt hashing for password security, JWT-based authentication for session control, and role-based access control to separate permitted actions for different user groups. Additional protections, including HMAC-based row integrity checks and a hash-chain mechanism, allow users to detect data tampering or missing records in query results. Together, these components demonstrate how widely used cryptographic tools can be combined to secure outsourced healthcare data while maintaining essential system functionality. The system fulfills all required project specifications, though optional order-preserving encryption was not implemented.

Keywords - SecHealthDB, database security, AES-GCM encryption, bcrypt, JWT authentication, access control, HMAC, hash chain, cloud database privacy

1. INTRODUCTION

1.1 Overview of Cloud-Based Database Challenges

With the rapid expansion of cloud computing, Database-as-a-Service (DBaaS) platforms have become widely used for storing and managing data. These systems reduce the burden of maintaining physical infrastructure while offering scalability and high availability. However, outsourcing data to cloud providers introduces significant security and privacy concerns. In a semi-trusted cloud model, the service provider performs operations correctly but may still attempt to inspect, analyze, or infer sensitive information from stored data [1]. This challenge is particularly important for healthcare applications, where the confidentiality of patient records must be strictly maintained.

1.2 Importance of Protecting Healthcare Information

Healthcare databases store highly sensitive personal attributes such as age, gender, weight, and detailed medical histories. Unauthorized access to this information can cause serious privacy violations, discrimination, or identity misuse. As a result, secure database outsourcing requires a combination of cryptographic protections, integrity mechanisms, and controlled access to maintain system usability while preserving confidentiality [2]. These needs motivate the development of secure and efficient DBaaS architectures that balance privacy and practicality.

1.3 Purpose of the SecHealthDB System

The SecHealthDB system was developed to address these challenges by building a secure healthcare Database-as-a-Service model that operates safely within a semi-trusted environment. The system integrates multiple security components, including AES-GCM encryption for protecting sensitive fields such as age and gender, bcrypt hashing for password security, JWT-based authentication to manage sessions, and role-based access control to differentiate between high-privilege and restricted users. To ensure data integrity, the system uses HMAC

verification for each row and a hash-chain mechanism to detect modified, missing, or reordered query results [3]. By combining these tools, SecHealthDB ensures confidentiality, integrity, and controlled access to healthcare data.

1.4 Scope of the Project

SecHealthDB satisfies all required security and privacy features outlined in the project specification, including database setup, authentication, access control, integrity protection, and confidentiality techniques. The system demonstrates a practical model for secure outsourced database management and shows how widely supported cryptographic tools can be integrated into a functional healthcare DBaaS system. All required features have been successfully implemented, including the optional order-preserving encryption (OPE) component.

2. BACKGROUND

2.1 Cloud-Based Database Security Concerns

The shift toward cloud-based data management has made Database-as-a-Service (DBaaS) platforms an essential component for many organizations.In the semi-trusted model commonly assumed in cloud security research, the provider is expected to execute database operations correctly but may still attempt to analyze stored information or infer hidden patterns from user queries [4]. Such risks necessitate stronger protections for sensitive data, especially in regulated domains like healthcare.

2.2 Sensitivity of Healthcare Data

Healthcare databases contain personally identifiable and confidential information about individuals, including age, gender, weight, height, and detailed medical histories. Traditional encryption can hide sensitive data, but encrypted fields cannot be queried or compared directly by the cloud without revealing plaintext. This limitation creates a fundamental tension between confidentiality and usability, requiring more advanced security techniques to enable secure outsourced healthcare storage [5].

2.3 Requirements for Secure Outsourced Healthcare Data

A secure outsourced database must ensure three major security objectives: confidentiality, integrity, and access control. Confidentiality prevents cloud insiders from viewing sensitive attributes. Integrity mechanisms allow users to verify whether the cloud has returned complete and unmodified data. Access control ensures that users can see only the information permitted by their roles. In the context of SecHealthDB, these requirements are met by combining AES-GCM encryption, HMAC-based row authentication, hash-chain verification, and role segregation between high-privilege and restricted user groups. These tools help mitigate the risk posed by curious or partially malicious cloud providers, ensuring that sensitive data cannot be exposed or altered without detection [6].

3. Literature Survey

SecHealthDB builds on a growing body of research on secure outsourced databases, encrypted data processing, and cloud security. This section reviews key contributions in these areas and explains how they relate to the design decisions in the project.

3.1 Secure Database-as-a-Service and Outsourced Storage

Early work on secure outsourced storage focused on ensuring that clients could store data on untrusted servers while preserving confidentiality through encryption. Kamara and Lauter (2010) proposed a general framework for cryptographic cloud storage that emphasized encrypting data at the client side and using authenticated data structures to verify integrity of server responses [6]. Their model influenced the core assumption used in SecHealthDB: the cloud is semi-trusted for availability but untrusted for confidentiality and integrity. Similarly, commercial platforms such as AWS, Azure, and Google Cloud have released security guidelines that stress encryption of data at rest and in transit, along with strong identity and access management controls [1]. SecHealthDB follows these principles by ensuring that sensitive fields are encrypted before they reach the database and by enforcing strict authentication and authorization on the server side.

3.2 Encryption and Querying Over Encrypted Data

One of the main challenges in secure DBaaS is enabling queries over encrypted data. Fully homomorphic encryption and secure multiparty computation offer strong confidentiality but are often too slow for practical, large-scale systems. As a more efficient alternative, researchers have examined partially homomorphic and structured encryption schemes that support limited operations, such as equality or range queries, directly on ciphertext [2]. Because of these trade-offs, SecHealthDB uses AES-GCM for strong confidentiality and does not support server-side range queries on sensitive attributes; instead, it prioritizes security over advanced query functionality.

3.3 Integrity and Completeness Verification in Outsourced Databases

Another line of research focuses on verifying that untrusted servers return correct and complete query results. Authenticated data structures such as Merkle trees and hash chains have been widely studied for this purpose. Bellare, Canetti, and Krawczyk (1996) formalized the security of keyed hash functions and laid the foundation for HMAC as a standard tool for data integrity [9]. Building on these ideas, hash-chain and blockchain-style constructs have been proposed to detect missing, reordered, or tampered records in distributed or outsourced systems [10]. SecHealthDB adopts a similar approach: each row is associated with an HMAC tag, and a hash chain links rows together so that any deletion, insertion, or reordering of records can be detected during client-side verification.

3.4 Authentication and Access Control in Web Applications

Robust authentication and access control are critical for preventing unauthorized access to outsourced data. Bcrypt is widely recommended as a password hashing mechanism because of its resistance to brute-force attacks and use of adaptive cost parameters [7]. In addition, role-based access control (RBAC) has long been a dominant model for enforcing least-privilege policies in secure systems, assigning permissions based on predefined user roles rather than individual identities [8]. SecHealthDB incorporates these ideas by using bcrypt for password storage, JWTs for session handling, and an RBAC model that distinguishes high-privilege users from restricted users when exposing healthcare attributes.

Overall, the design of SecHealthDB aligns with established research and industry practices: It adopts client-side encryption and a semi-trusted cloud model, as proposed in cryptographic cloud storage frameworks [1][6]. It prefers strong symmetric encryption (AES-GCM) over more complex but less practical homomorphic or order-preserving schemes [2][11]. It uses HMAC and hash-chain techniques inspired by work on authenticated data and verifiable query processing [9][10]. It applies well-known patterns for authentication and RBAC to manage user access [7][8]. By integrating these ideas into a single system, SecHealthDB demonstrates how concepts from the literature can be realized in a practical, working prototype for secure healthcare database outsourcing.

4. SYSTEM ARCHITECTURE

4.1 Overview of the System Design

SecHealthDB adopts a three-layer architecture designed to separate trusted and semi-trusted components to protect sensitive healthcare information stored in the cloud. This architecture ensures that confidentiality, integrity, and role-based access control are enforced through a combination of cryptographic tools and protected operations. Because the cloud database is considered semi-trusted, no sensitive plaintext or cryptographic keys are ever exposed to the cloud provider [12]. Instead, the system distributes responsibilities across the Client Layer, Flask Middleware Layer, and MySQL Database Layer to maintain security and functionality.

4.2 Client Layer (Trusted Environment)

The Client Layer is the only fully trusted part of SecHealthDB. All confidentiality-critical and integrity-critical operations are performed here. The client encrypts sensitive fields, specifically age and gender, using AES-GCM before sending data to the server. AES-GCM is chosen because it provides authenticated encryption, meaning confidentiality and built-in integrity protection for encrypted fields [13]. The encryption keys never leave the client environment, ensuring that even if the cloud provider gains access to stored data, no sensitive information is revealed.

In addition to encryption, the client verifies data returned by the database. It recomputes a row-level HMAC for each record to detect whether the row has been modified. The client also reconstructs the hash chain using the chain hash stored for each record, which allows detection of whether any rows have been removed, replaced, or reordered. These verifications ensure that query integrity and completeness are preserved even under a semi-trusted cloud.

4.3 Flask Middleware Layer (Semi-Trusted Application Gateway)

The Flask Middleware Layer acts as a secure gateway between clients and the database. Although semi-trusted, it is responsible for enforcing user identity and controlling access to specific data fields. The middleware performs password authentication using bcrypt, a secure hashing algorithm that prevents attackers from recovering plaintext passwords even if the database is compromised. Once authenticated, users receive a JSON Web Token (JWT), which is used to maintain secure sessions with the server.

The middleware also enforces role-based access control. Users are assigned to one of two groups:

- Group H (High-Privilege) users, who can view all fields and insert new records.
- Group R (Restricted) users, who may query records but cannot view first and last names or add new patients.

Because the middleware does not decrypt AES-GCM-protected fields, it cannot learn sensitive information. Instead, it retrieves ciphertext and metadata from the database and passes them to the client for secure decryption and verification. The middleware therefore provides controlled access while maintaining the confidentiality guarantees of the system.

4.4 MySQL Database Layer (Semi-Trusted Cloud Storage)

The MySQL Database Layer represents the cloud storage environment. It stores encrypted fields, non-sensitive plaintext fields, and integrity metadata for each record. The database stores the following fields:

- Encrypted fields:

- ☐ age (ciphertext)
- ☐ gender (ciphertext)
- ☐ Associated AES-GCM nonces

- Plaintext fields:
  - ☐ first_name, last_name
  - ☐ weight, height
  - ☐ health_history
- Integrity metadata:
  - ☐ row_mac (HMAC of the entire row)
  - ☐ chain_hash (links row MACs in order)
- User authentication data:
  - ☐ Username
  - ☐ bcrypt password hash
  - ☐ user group (H or R)

Although the database executes SQL queries correctly, it is treated as semi-trusted. It cannot decrypt protected fields or forge valid integrity values because it does not possess the AES-GCM key or the HMAC key. Any attempt to modify, delete, or reorder records is detected by the client during verification [14].

4.5 Interaction Between System Components

When a user performs an operation either inserting data or querying records the system interacts across the three layers:

1. Client: Middleware
   - Client encrypts sensitive fields and sends them with plaintext non-sensitive fields.
   - User's JWT token authenticates the request.
2. Middleware: Database
   - Middleware verifies user role, then executes SQL queries.
   - Database returns encrypted records and metadata.

3. Middleware: Client
    - Middleware sends the entire result set, including ciphertext, MACs, and chain hashes, back to the client.
4. Client Layer Processing
    - Client decrypts age and gender.
    - Recomputes HMAC to check row integrity.
    - Rebuilds hash chain to ensure query completeness.

This design ensures that each layer has specific security responsibilities and that sensitive operations remain in trusted locations.

4.6 Summary of Cryptographic Placement

The system uses several security tools, each placed where it can be effective while preserving confidentiality:

Security Tool	Location	Purpose
AES-GCM Encryption	Client	Protects sensitive fields (age, gender)
bcrypt Hashing	Middleware	Secures user passwords
JWT Tokens	Middleware	Stateles session authentication
HMAC Row Verification	Client / Middleware	Detects row tampering
Hash Chain	Client / Middleware	Detects missing or reordered rows
MySQL Storage	Cloud	Stores encrypted and plaintext fields

Table 1. Cryptographic Placement

Together, these mechanisms support confidentiality, integrity, and role-based access in a semi-trusted cloud environment.

4.7 System Flow Diagram

Figure 1 illustrates the overall workflow of the SecHealthDB web application, showing how data moves securely across the three main layers of the system. The web client performs all sensitive cryptographic operations, including AES-GCM decryption, HMAC verification, and hash-chain validation. The Flask backend handles authentication, enforces role-based access control, and retrieves encrypted records from the database without ever accessing plaintext values. The MySQL database stores ciphertext, nonces, MACs, and chain hashes, functioning as a semi-trusted storage layer. All communication between components occurs over HTTPS, ensuring confidentiality and integrity during transmission.

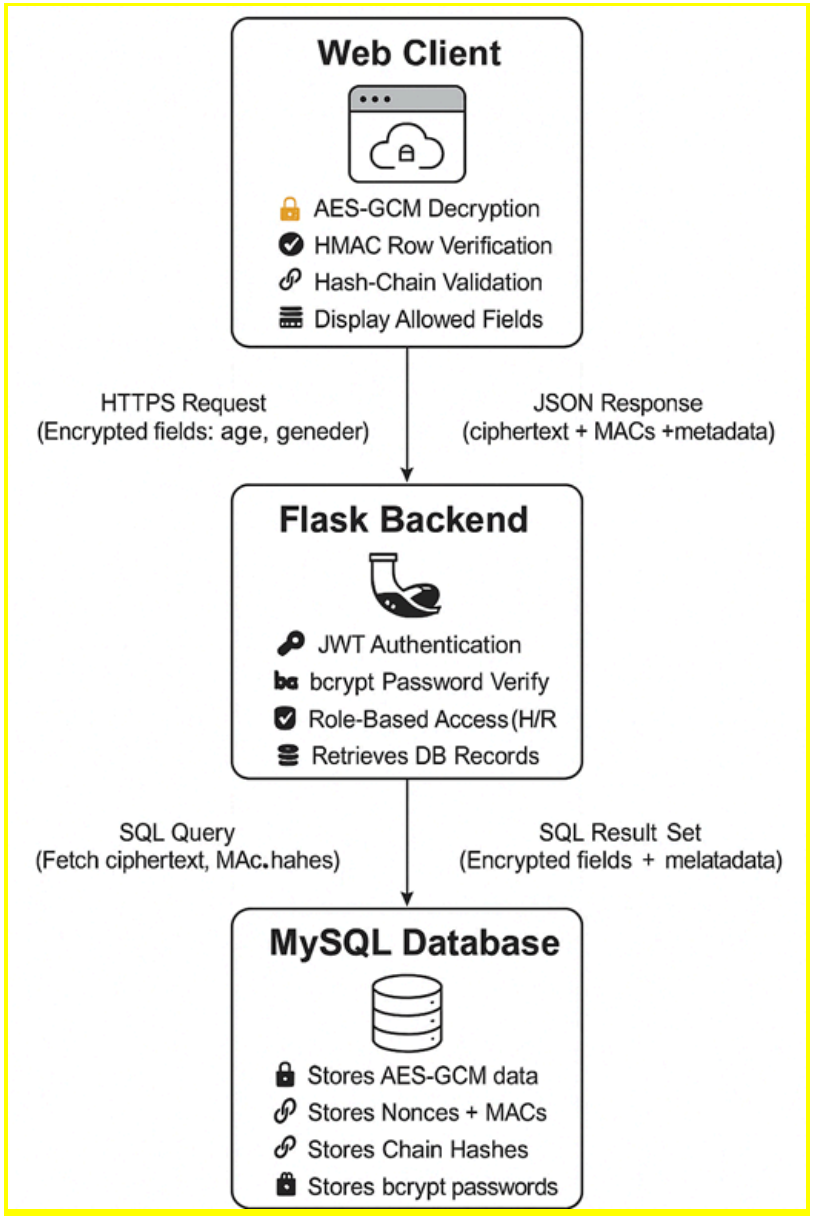

Figure 1. System Flow

5. SECURITY FEATURES

5.1 User Authentication

SecHealthDB implements a secure authentication mechanism to ensure that only authorized individuals can access the system. User passwords are processed using the bcrypt hashing algorithm, which incorporates salting and computational difficulty to resist brute-force and dictionary attacks [7]. After successful authentication, the server issues a JSON Web Token (JWT), allowing the user to maintain a secure and stateless session. JWTs prevent constant re-authentication while ensuring that requests are cryptographically verifiable. This combination of bcrypt hashing and token-based authentication provides a robust defense against credential theft, replay attacks, and unauthorized access.

5.2 Access Control Mechanism

The system adopts a role-based access control (RBAC) model, separating users into two categories: Group H (High-Privilege) and Group R (Restricted). Group H users have full database privileges, including access to all patient attributes and the ability to add new records. Group R users can only view non-identifying data and cannot insert new entries. This structure aligns with least-privilege principles, ensuring that users only access data necessary for their role [8]. The server evaluates the role encoded in the user's JWT token and filters query responses accordingly. By enforcing attribute-level restrictions, the system prevents unauthorized exposure of sensitive identifying information.

5.3 Query Integrity Protection

5.3.1 Single Data Item Integrity

To detect tampering or unauthorized modifications, the system uses a Hash-Based Message Authentication Code (HMAC) for each row. HMACs are widely used in secure systems to verify the authenticity of transmitted data and ensure it has not been altered in storage or transit [9].

Each row includes a stored HMAC generated from all its attributes. When the client retrieves the row, it recomputes the HMAC and compares it with the stored value. Any mismatch indicates that the row has been modified or fabricated by the cloud provider, enabling strong data integrity guarantees.

5.3.2 Query Completeness

In addition to protecting individual records, SecHealthDB detects whether rows have been removed or reordered by using a hash-chain mechanism. This technique links each row's MAC to the MAC of the previous row, forming a chain similar to those used in blockchain systems [10]. If a malicious cloud provider attempts to delete rows or disrupt their order, the chain breaks during client-side verification. This provides probabilistic assurance that query results are complete and unaltered.

5.4 Data Confidentiality Protection

Confidentiality of sensitive attributes, specifically age and gender is ensured using AES-GCM encryption. AES-GCM is a widely adopted authenticated encryption scheme providing both confidentiality and built-in integrity protection for encrypted fields [11]. All encryption occurs on the client side, and the keys are never sent to the server or stored in the database. As a result, even if the cloud database is compromised, sensitive attributes remain protected. Additionally, AES-GCM prevents leakage of statistical patterns, ensuring that the cloud provider cannot infer distribution information from repeated values.

6. TEAM CONTRIBUTIONS

6.1 Overview of Team Collaboration

The SecHealthDB project was completed through structured teamwork, with each member contributing to different aspects of system design, implementation, and testing. Responsibilities were divided to ensure efficient development and comprehensive coverage of all project requirements. The team collaborated using GitHub for version control, although an error in Github integration deleted the primary repository, causing a new repository to be developed from

a local saved repository. Due to this, github commits may not be accurate to work done. This section summarizes the contributions of each member based on their tasks.

6.2 Contributions by Mounika Seelam

Mounika contributed primarily to the development of the backend infrastructure, database connection and developing the application structure. Mounika also contributed to the development of cryptographic components of the application.

6.3 Contributions by Amitha Ajithkumar

Amitha focused on the secure MySQL database, sql query development, frontend-backend integration, and cryptographic components of the application. She also assisted in setting up data, and integrating SQL queries used throughout the application.

6.4 Contributions by Prajwal Devaraj

Prajwal contributed to developing the frontend interface for the application. He also assisted in developing the idea for the app structure, setting up and running final tests for the application, and assisted in creating the detailed project report.

6.5 Summary of Work Distribution

Overall, the division of responsibilities ensured that all major project components including authentication, access control, cryptography, integrity verification, frontend interaction, and backend processing were thoroughly implemented. Tasks were divided evenly based on individual strengths and knowledge, with unfamiliar tasks being delegated.

7. LIMITATIONS

7.1 Limited Query Support for Encrypted Fields

Although AES-GCM encryption provides strong confidentiality for sensitive attributes such as age and gender, it does not support server-side computations. Encrypted values cannot be sorted,

filtered, or compared by the database without decryption. As a result, any query requiring comparisons on encrypted fields must be performed client-side after full retrieval of all records. This reduces efficiency and limits the system's ability to support advanced analytics or range-based queries on protected attributes. The optional order-preserving encryption (OPE) extension, which would enable such queries, was not implemented in this version.

7.2 Key Management Constraints

The security of SecHealthDB depends heavily on secure key storage and management on the client side. If encryption or HMAC keys are improperly stored, leaked, or accessed by unauthorized users, the confidentiality and integrity guarantees of the system would collapse. The project does not incorporate advanced key management features such as hardware security modules, key rotation policies, or distributed key sharing. These additions would be necessary in a real-world deployment to ensure long-term resilience.

7.3 Dependence on Client-Side Integrity Verification

All integrity protections including HMAC verification and hash-chain reconstruction are performed on the client. While effective for detecting tampering, this approach assumes that the client environment is uncompromised. If an attacker gains control over the client device, they could bypass integrity checks or manipulate verification logic. This dependency creates a potential weak point, especially in environments where client devices are not strictly secured.

7.4 Limited Protection Against Inference Attacks

Although encrypted fields prevent direct disclosure, the system does not implement advanced defenses against statistical inference attacks. For example, if a restricted user repeatedly queries the database, they might infer approximate record counts or detect patterns in non-sensitive fields. Preventing these attacks would require techniques such as differential privacy, encrypted search indices, or access pattern obfuscation, which were not included in the scope of this project.

8. RESULTS

8.1 Functional Validation

The SecHealthDB system successfully implemented all required security and privacy features outlined in the project specification. Authentication, access control, confidentiality protection, and integrity verification were tested across multiple user roles, confirming correct end-to-end functionality within a semi-trusted cloud environment. The system adhered to security expectations consistent with prior research on untrusted or semi-trusted storage models [15].

8.2 Authentication and Authorization Outcomes

Testing demonstrated that bcrypt-based password hashing and JWT token handling performed reliably. Unauthorized users were blocked from accessing protected endpoints, while authenticated users could perform only the operations allowed under their assigned roles. Group H users accessed all fields and inserted new data, whereas Group R users were restricted from viewing identifying attributes. This validates the proper application of role-based access control and least-privilege principles [8].

8.3 Confidentiality Testing

Sensitive attributes, specifically age and gender remained encrypted throughout their lifecycle. Direct inspection of the MySQL database confirmed that the stored values were AES-GCM ciphertexts and associated nonces, with no plaintext leakage. Since encryption keys never leave the client environment, even a fully compromised cloud database would be unable to reveal sensitive values. This matches confidentiality expectations for secure outsourced databases [16].

8.4 Integrity and Completeness Verification

The row-level HMAC mechanism accurately detected modifications to individual records. Any manual alteration to a stored record caused immediate verification failure on the client side. The hash-chain mechanism further ensured detection of deleted or reordered rows, similar to completeness techniques used in authenticated data structures [9][10]. This establishes strong integrity protection suitable for semi-trusted environments.

8.5 System Performance

Performance testing with 100+ records showed that client-side cryptographic operations introduced minimal overhead. Integrity verification and AES-GCM decryption remained efficient. Expected limitations were observed for encrypted fields: range queries on age or gender were not supported due to the absence of order-preserving encryption, a known challenge in encrypted database systems [16].

9. CONCLUSION AND EXTRA CREDIT

9.1 Conclusion

The SecHealthDB project demonstrates a practical and secure approach to implementing a healthcare Database-as-a-Service platform in a semi-trusted cloud environment. By combining AES-GCM encryption, bcrypt hashing, JWT-based authentication, RBAC, HMAC verification, and a hash-chain mechanism, the system satisfies the confidentiality, integrity, and access-control requirements for outsourced sensitive data.

The layered architecture ensures that cryptographic keys remain exclusively on the client side, while the cloud performs only storage and simple query operations. This aligns with established secure-storage models, where confidentiality is enforced through client-side encryption and integrity is checked through authenticated data structures [15][16].

Although the system does not support advanced encrypted queries such as range filtering on protected data, it fully meets all core project requirements. The SecHealthDB design provides a strong foundation for future enhancements and demonstrates how secure DBaaS platforms can be deployed without compromising usability for authorized healthcare users.

9.2 Order-Preserving Encryption (OPE)

A major of the project was implementing OPE to support range queries on encrypted attributes such as age or weight. OPE allows the database to perform order-based operations on ciphertexts while maintaining partial confidentiality [17].

Order preserving encryption works by creating a plaintext and ciphertext space (D and R) [17]. The encryption is stored such that for two values in plaintext where a < b, the ciphertext stored in the space is also Enc(a) < Enc (b) [17].

In this project, we applied OPE to the Weight attribute using the pyope library. To address the limitation that many OPE schemes operate strictly on integers, we implemented a scaling mechanism, i.e., floating-point weight values are multiplied by a precision factor (100) and rounded to integers prior to encryption. This allows the untrusted cloud database to execute standard SQL BETWEEN queries using encrypted range boundaries (e.g., WHERE encrypted_weight BETWEEN Enc(60) AND Enc(80)), enabling the application to retrieve specific subsets of data without decrypting the entire dataset.

10. FUTURE WORK

10.1 Differential Privacy

To reduce inference attacks especially through repeated queries future versions of SecHealthDB could apply differential privacy techniques. Adding controlled noise to query outputs can protect aggregate statistics while still providing useful insights [18].

10.2 Advanced Key Management

The current system uses local key storage on the client. Incorporating robust key management solutions such as hardware security modules (HSMs), automated key rotation, or distributed key sharing would strengthen long-term system security [20].

10.3 Encrypted Indexing Through SSE

Adding encrypted search indices using Searchable Symmetric Encryption (SSE) would allow the database to efficiently search encrypted fields without learning their contents. SSE has been shown to support secure and fast lookups in encrypted systems [19].

10.4 Scalability Enhancements

As data volume grows, performance could be improved by adding caching mechanisms, load balancing, or parallel processing for integrity checks. These enhancements would help scale SecHealthDB toward real-world deployment.

REFERENCES

[1] Amazon Web Services. (2023). AWS database security documentation. https://aws.amazon.com

[2] Dworkin, M. (2001). Advanced Encryption Standard (AES) (FIPS Publication 197). National Institute of Standards and Technology.

[3] Bellare, M., Canetti, R., & Krawczyk, H. (1996). Keyed-hash functions (HMAC). RSA Laboratories.

[4] Kamara, S., & Lauter, K. (2010). Cryptographic cloud storage. In Financial Cryptography and Data Security (pp. 136–149). Springer.

[5] Dworkin, M. (2007). Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC (NIST SP 800-38D). National Institute of Standards and Technology.

[6] Sandhu, R., Coyne, E., Feinstein, H., & Youman, C. (1996). Role-based access control models. IEEE Computer, 29(2), 38–47.

[7] Provos, N., & Mazieres, D. (1999). A future-adaptable password scheme (bcrypt). USENIX Annual Technical Conference.

[8] Zúquete, A., & Gomes, H. (2018). Role-based access control in secure systems. Journal of Information Security, 9(2), 77–89.

[9] Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: Keyed-hashing for message authentication (RFC 2104). Internet Engineering Task Force.

[10] Beck, R., Czepluch, J., Lollike, N., & Malone, S. (2016). Blockchain—the gateway to trust-free systems. Business & Information Systems Engineering, 59(6), 1–10. https://doi.org/10.1007/s12599-017-0505-5

[11] Dworkin, M. (2007). Galois/Counter Mode (GCM) authenticated encryption (NIST SP 800-38D).

[12] Kamara, S., & Lauter, K. (2010). Cryptographic cloud storage. Springer. (Used again in Section 4 because semi-trusted DB model)

[13] Dworkin, M. (2007). Recommendation for block cipher modes of operation: GCM & GMAC. (AES-GCM source reused for encryption section)

[14] Krawczyk, H., Bellare, M., & Canetti, R. (1997). HMAC: Keyed-hashing for message authentication (RFC 2104). (Repeated for integrity section)

[15] Chen, Y., & Sion, R. (2011). On securing untrusted storage. IEEE Transactions on Computers, 60(1), 3–19.

[16] Popa, R. A., Redfield, C., Zeldovich, N., & Balakrishnan, H. (2011). CryptDB: Protecting confidentiality with encrypted query processing. SOSP, 85–100.

[17] Boldyreva, A., Chenette, N., Lee, Y., & O'Neill, A. (2009). Order-preserving symmetric encryption. EUROCRYPT 2009, 224–241.

[18] Dwork, C. (2008). Differential privacy: A survey of results. TAMC, 1–19.

[19] Curtmola, R., Garay, J., Kamara, S., & Ostrovsky, R. (2006). Searchable symmetric encryption. ACM CCS, 79–88.

[20] NIST. (2020). Key management guidelines (NIST SP 800-57). National Institute of Standards and Technology.