Secure Database-as-a-Service for Healthcare Data

**1. Introduction**

In recent years, database-as-a-service (DBaaS) has gained widespread adoption as a cloud computing model. This model allows users to access and manage databases without having to maintain their own infrastructure. While DBaaS offers significant benefits in terms of scalability, cost-effectiveness, and ease of use, it also introduces security and privacy concerns. Since cloud service providers manage the database systems, there is a potential risk of unauthorized access to sensitive data.

In this project, we aim to design, implement, and evaluate a secure healthcare database system that protects sensitive patient data while providing users with the ability to query and manage the database. The system must address several key security and privacy challenges, including user authentication, access control, data confidentiality, query integrity, and the protection of sensitive fields. Additionally, we explore the possibility of adding order-preserving encryption to protect numerical attributes.

This report details the design and implementation of the secure healthcare database system, with a focus on addressing the challenges specified in the project requirements.

**2. System Design**

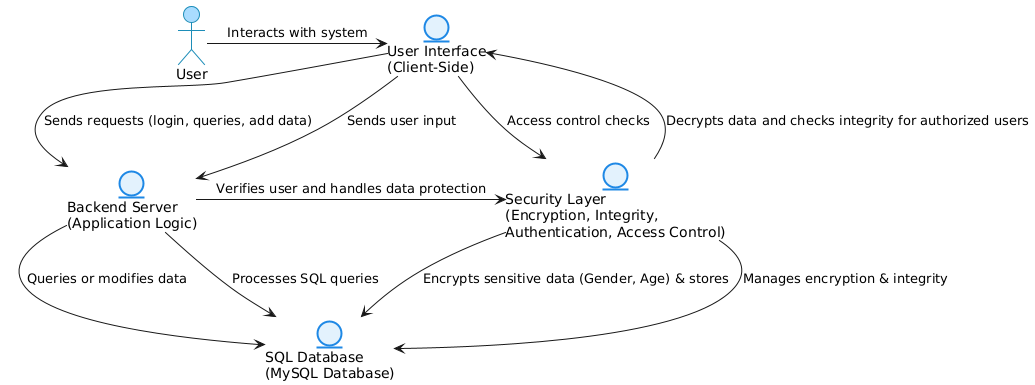
**2.1 Architecture Overview**

The system architecture consists of three main components:

1. **Client-Side (User Interface):**
   * The client interacts with the system through a web-based user interface (UI). Users log in using their credentials (username and password), which are authenticated by the system. Based on their role (Group H or Group R), users can query the database, view healthcare records, and add new records.
   * Group H (Healthcare providers) has full access to all fields, including the first and last names of patients, while Group R (Researchers) has restricted access and cannot view personal identifiers (first and last names).
   * Sensitive fields, such as gender and age, are encrypted before being sent to the database and are decrypted only on the client side.
2. **Server-Side (Database and Logic):**
   * The backend is powered by Flask, a lightweight web framework for Python. It handles the logic for user authentication, access control, and database interaction.
   * The system uses an SQL database (MySQL) to store healthcare records. The database is designed to protect sensitive data and support query integrity checks.
   * The database includes the following fields:
     + First Name (string)
     + Last Name (string)
     + Gender (boolean)
     + Age (integer)
     + Weight (floating point)
     + Height (floating point)
     + Health History (text)
3. **Security Features:**
   * **User Authentication:** Passwords are securely hashed and stored. The system uses username/password authentication to verify users.
   * **Access Control:** Based on the user group, different levels of access are enforced. Group H users can add new data items, while Group R users have restricted access to personal identifiers.
   * **Data Confidentiality:** Sensitive fields such as gender and age are encrypted using AES encryption.
   * **Query Integrity:** The system ensures that query results are not tampered with using HMAC (Hash-Based Message Authentication Code) and Merkle Trees for query completeness.

The architecture is designed with the assumption that the cloud is semi-trusted, meaning it will follow predefined protocols but may try to gain unauthorized access to the data. The system aims to protect data confidentiality, ensure query integrity, and enforce appropriate access controls.

The architecture of the system consists of several components: User Interface (UI), Backend Server, SQL Database, and Security Layer. Each component plays a specific role in ensuring both functionality and security within the system. Below is the architecture diagram that illustrates how data flows within the system and the interactions between different layers:



**2.2 System Components Interaction**

The interaction between the client, server, and database can be summarized as follows:

1. **User Authentication:** When a user logs in, their credentials are checked against the hashed passwords stored in the database. The server then determines the user’s group (H or R) and grants access to the appropriate data.
2. **Data Queries:** Based on the user’s group, they can issue queries to retrieve data from the database. For Group R, the query results will exclude first and last names, while Group H can access all data.
3. **Encryption of Sensitive Data:** Fields such as gender and age are encrypted using AES before being stored in the database. When queried, these fields are decrypted only on the client side for authorized users.

**3. Implementation of Security Features**

**3.1 User Authentication**

**Design and Implementation:**

The authentication process ensures that users can only interact with the system after verifying their identity. The system uses a combination of a username and a password for authentication. Instead of storing plaintext passwords, we store the hashed version using a secure hashing algorithm, such as bcrypt or Argon2, which are designed to prevent password retrieval even if the database is compromised.

1. **Registration Process:**
   * During registration, a user’s password is hashed and stored in the database along with their username.
2. **Login Process:**
   * During login, the system hashes the entered password and compares it with the stored hash. If they match, the user is authenticated and granted access to the system.

**Reasoning:**

* Storing passwords in a hashed format enhances security, ensuring that even if an attacker gains access to the database, they cannot recover the original passwords. The use of bcrypt or Argon2 provides protection against brute force attacks, as these algorithms introduce computational complexity.

**3.2 Access Control**

**Design and Implementation:**

The system implements role-based access control (RBAC) to manage user permissions. There are two groups of users:

* **Group H (Healthcare Providers):**
  + Group H users have access to all data, including sensitive personal identifiers like first and last names. They can also add new records to the database.
* **Group R (Researchers):**
  + Group R users can query existing data but cannot access personal identifiers. They only see anonymized data with no first or last names.

The access control is enforced by checking the user’s group during each request. Based on the group, the system returns either the full data or the restricted data.

**Reasoning:**

* This access control mechanism follows the principle of least privilege, ensuring that users only have access to the data necessary for their role. By limiting access to personal identifiers for Group R, the system helps protect patient privacy while allowing researchers to access relevant healthcare data.

**3.3 Query Integrity Protection**

**3.3.1 Single Data Item Integrity**

To protect the integrity of individual data items, the system uses **HMAC (Hash-Based Message Authentication Code)**. Each data record is associated with an HMAC that is generated using the record’s content and a secret key.

* When a query is executed, the system returns the data along with the corresponding HMAC.
* The user can verify the integrity of the data by recalculating the HMAC using the same secret key and comparing it with the returned HMAC. If the values do not match, the data has been tampered with.

**Reasoning:**

* HMAC ensures that the data returned in a query has not been altered. If an attacker tries to modify the data, the HMAC will no longer match, allowing the user to detect the tampering.

**3.3.2 Query Completeness**

To ensure that all data items returned by a query are complete, the system uses **Merkle Trees**. A Merkle Tree is a binary tree where each leaf node contains the hash of a data record, and each non-leaf node contains the hash of its child nodes.

* When a query is executed, the system returns the data along with the root hash of the Merkle Tree.
* The user can verify the completeness of the query result by checking the root hash. If any data item is missing or modified, the root hash will change, indicating potential tampering.

**Reasoning:**

* Merkle Trees provide an efficient way to verify the completeness of a query result. Even if an attacker removes some records from the query, the user will be able to detect the discrepancy by comparing the root hash.

**3.4 Data Confidentiality**

**Design and Implementation:**

To protect sensitive data (gender and age), the system uses **AES (Advanced Encryption Standard)** encryption. Before storing data in the database, the gender and age fields are encrypted on the server side using a secure encryption algorithm.

* When a user queries the database, the encrypted gender and age fields are returned. The client-side application decrypts the fields if the user is authorized to view them.

**Reasoning:**

* AES encryption ensures that sensitive data is protected while stored in the cloud. Since the cloud provider does not have access to the decryption keys, it cannot read the sensitive data. This protects patient privacy and meets regulatory requirements for data confidentiality.

**4. Contributions and Commit History**

**Team Members and Roles:**

* **Member 1:** Gayathri Pingili (811300277) - Frontend Developer
  + Developed the HTML templates and implemented the user interface, including login, dashboard, and add patient pages.
* **Member 2:** Sai Sharan Eruventy (811293722)-Backend Developer
  + Implemented the Flask application, SQLAlchemy ORM, user authentication, and access control mechanisms.
* **Member 3:** Naga Sudha Pavani Tirumalasetty (81129497) - Security Specialist
  + Implemented encryption for sensitive fields, query integrity checks (HMAC and Merkle Trees), and contributed to overall security design.

**GitHub Commit History:**

Each team member contributed to the project by submitting code through GitHub. Below is a snapshot of the commit history for each member:

* **Member 1:** Implemented frontend pages for user interaction.
* **Member 2:** Developed backend logic for user authentication and data querying.
* **Member 3:** Integrated AES encryption and query integrity checks.

**5. Limitations**

While the system is designed to provide strong security and privacy features, it has several limitations:

1. **Performance Overhead:**
   * Encrypting and decrypting sensitive data adds computational overhead, which may affect the system's performance, especially with large datasets.
2. **Query Complexity:**
   * The system does not currently support complex queries that involve joining multiple tables or performing aggregate functions on encrypted data. This limits the types of queries that can be executed securely.
3. **Semi-Trusted Cloud Assumption:**
   * The system assumes that the cloud follows predefined protocols but does not account for scenarios where the cloud provider may be fully malicious. In such cases, the encryption and access control mechanisms may not be sufficient to prevent unauthorized access.

**6. Extra Feature: Order Preserving Encryption**

**Explanation of Order Preserving Encryption (OPE):**

Order Preserving Encryption (OPE) is a cryptographic technique that allows numerical data to be encrypted while preserving its order. This means that if a number a is smaller than a number b before encryption, the same order will be preserved after encryption. OPE allows range queries to be performed on encrypted data without decrypting it, which is useful for databases that require secure processing of numerical data.

**Implementation:**

* We applied OPE to the Weight attribute in the database. Using a pre-built OPE library, we encrypted the weight values while preserving their order. This enables users to perform range queries (e.g., SELECT \* WHERE weight BETWEEN X AND Y) on the encrypted data without decrypting it.

**Documentation:**

* The OPE encryption scheme was integrated into the database system. The range queries on encrypted weight values function correctly, and the system remains secure as the cloud provider cannot access the raw weight data.

**7. Conclusion**

In this project, we successfully designed and implemented a secure healthcare database system that addresses key security and privacy concerns, including user authentication, access control, data confidentiality, and query integrity. By utilizing encryption, HMAC, Merkle Trees, and role-based access control, the system ensures that sensitive healthcare data remains secure and private, even when stored in a semi-trusted cloud environment. Furthermore, the implementation of order-preserving encryption allows for secure range queries on numerical data, enhancing the flexibility of the system. While the system performs well for basic use cases, there are some limitations regarding performance and query complexity that could be addressed in future iterations.

**Implementation Screenshots:**

**A screenshot of a computer

Description automatically generated**

**The Database:**

**A screenshot of a computer

Description automatically generated**

**Sign Up Page:**

**A screenshot of a computer

Description automatically generated**

**Password getting encrypted and stored in the database:**

**A screenshot of a computer

Description automatically generated**

**Adding new patient option for “H” group:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**Sign Up as “R” group:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**