**Project Report:****Secure Database-as-a-Service System.**

**Team 6: Priyanka Vyas**

**1.Design Overview and System Architecture Explanation:**

The implemented code represents a health record management system designed to store, manage, and secure patient data. The system architecture is composed of several key components, each playing a critical role in ensuring the functionality, security, and privacy of the data handled. The diagram below reflects the flow of data and control across key components, from the user making a query or entering data, through authentication and access control, to the storage and retrieval of data in the database, with a focus on security and privacy features.

A diagram of data processing

Description automatically generated

The setup, as described in the source code, meets the basic requirements stated: using a SQL database (MySQL) and running it on a personal computer while simulating a cloud environment. The system utilizes a MySQL database for data storage and retrieval. This aligns with the requirement of using a SQL database. The source code indicates a local setup (i.e. on a personal computer), where the MySQL database is hosted locally. This setup fits the criteria of running the database on a personal computer. The aspect of "pretending" to be in a cloud environment can be interpreted as simulating aspects of cloud functionality (like remote access, scalability, or security features) in a local setup. While the code doesn't explicitly simulate a cloud environment, it doesn't preclude such a setup either. Below is a detailed explanation of the system architecture aligned with the implemented code:

**1. User Interface Layer**

**Main Interface:** This is the point of interaction with users. It includes options for user registration, login, and role-specific operations like viewing, adding, or modifying patient data.

**Role-Based Actions:** Depending on the user's role (admin or user), different functionalities are accessible. Admins have broader access, including adding patient data and performing tests for security features.

**2. Authentication and Authorization**

**User Authentication**: Utilizes bcrypt for hashing passwords, ensuring that actual passwords are never stored in the database, enhancing security against unauthorized access. It handles user login using username/password, without storing the original password.

**Role-Based Access Control**: Determines user group (H or R) and controls access to fields in the database. The system checks the user's role to determine the level of access to patient data, implementing the principle of least privilege.

**3. Database Layer**

**MySQL Database**: Stores healthcare data. This includes fields like First name, Last name, Gender, Age, Weight, Height, Health history. This is where all the patient data, user credentials, and other relevant information are stored. Manages interactions with the SQL database, applies access control and data confidentiality rules.

**Patient Data Management**: Functions for adding, retrieving, and managing patient data are implemented, with security measures such as encryption and data integrity checks.

**4. Data Encryption and Integrity**

**AES Encryption**: Encrypts sensitive fields (Gender and Age) for confidentiality. Sensitive patient data (like gender and age) is encrypted using AES encryption, which is a robust standard for securing data.

**Data Integrity Checks**: Processes queries from users, checks for completeness and integrity of query results. SHA-256 hashing is used to generate a unique hash for each patient record, allowing the system to detect any unauthorized changes to the data.

**5. Security Utilities**

**Encryption Key Generation**: The system generates AES keys for encryption purposes.

**Data Padding**: Sensitive data is padded with random characters before encryption to add an additional layer of security.

**Hash Generation**: Used for ensuring data integrity by comparing the generated hash of the data with the stored hash.

A computer screen shot of a data base

Description automatically generated

**Above diagram specifically explains the System Flow for easy reference:**

**User Interaction**: Users interact with the system through a command-line interface, choosing options like login, register, or perform role-specific tasks.

**Authentication and Role Determination**: User credentials are verified, and roles are determined, dictating the level of access.

**Data Processing**: Based on user input and role, patient data is either retrieved, added, or modified. This involves encryption/decryption for sensitive fields and data integrity checks.

**Database Interaction**: All interactions with patient data are conducted through SQL queries to the MySQL database.

**Security Operations**: Throughout this process, security features like hashing, encryption, and access control are consistently enforced.

In the existing system, the Application Layer (handling user interaction and data processing) would communicate with the Database (hosted in locally, in this scenario) through the Network Connection/layer. If simulating a cloud environment, even when operating locally, the Network Layer would manage data flow as if it were communicating with remote servers (akin to a cloud setup), ensuring secure data transmission.

**2.Project Requirement checks in line with source code:**

**2.1.Database Setup and Table Creation**

**Database Creation**:

The system first establishes a connection to MySQL using ‘pymysql.connect’, connecting to the MySQL server as the root user. It then creates a new database named ‘healthdetailsdb’ using the SQL command ‘CREATE DATABASE IF NOT EXISTS ‘healthdetailsdb’.

**A screen shot of a computer

Description automatically generated**

**Table Creation**:

There are two tables created: ‘user\_credentials’ and ‘patient\_info’. The ‘patient\_info’ table is of particular interest as it stores patient data. This table includes columns for first name, last name, gender, age, weight, height, and health history, along with other fields related to security (like encrypted data and keys). The ‘populate\_patient\_data’ function is used to generate and insert patient data into the ‘patient\_info’ table. It utilizes the Faker library to generate fake data for patient details like names, and uses ‘random’ for age, weight, and height. The function creates a dictionary for each patient record with the required fields. The gender and age fields are encrypted before insertion (for security purposes). A unique hash (‘record\_hash’) is generated for each record for data integrity checks. The SQL ‘INSERT INTO’ command is used to insert each patient record into the ‘patient\_info’ table. The function is designed to insert a specified number of records (‘num\_records=100’) into the database. This is achieved by looping ‘num\_records’ times, each time generating a new patient record and inserting it into the database. A specific function, ‘validate\_patient\_data’, is used to validate the patient data before it is entered into the database. This function includes checks to ensure that certain data fields do not have negative values.

A screenshot of a computer code

Description automatically generated

**2.2. Security Features Implementation, Testing and Evaluation:**

For each security feature implemented as per Section 4.2, below is the explanation about the way it is implemented and the reason it achieves the specific security feature:

**1. User Authentication Requirements**: Username/password-based authentication, with the cloud not storing original passwords.

**Implementation**: User passwords are hashed using bcrypt before being stored in the database. Authentication is performed by comparing the stored hash with the hash of the provided password.

* **Password Hashing**: The system uses bcrypt to hash passwords before storing them in the database. bcrypt is a reliable cryptographic algorithm known for its security and resistance to brute-force attacks. Hashed passwords are secure and irreversible, meaning the original password is not stored or recoverable.
* **Registration and Login Process**: When a new user is registered, their password is hashed and then stored. During login, the system retrieves the stored hash and compares it with the hash of the provided password for authentication.

**Effectiveness:** This protects against password theft, as the actual passwords are not stored, and bcrypt is resistant to brute-force attacks due to its computational cost.

**Testing Scenario**:

* **Register a New User**: Create a new user and observe the hashed password stored in the database.
* **Login Attempts**: Attempt to log in with both correct and incorrect passwords, validating that authentication only succeeds with the correct password.

A screenshot of a computer

Description automatically generated

The screenshot shows the successful login and Login failed using the correct and incorrect password for an already created UserR

**2. Basic Role-Access Control Mechanism Requirements**: Admins (Group H) access all fields; Users (Group R) access all except first name and last name.

**Implementation**: Users are assigned roles (admin or user), and access to data is controlled based on these roles.

* **Role Assignment**: Upon registration, users are assigned roles (admin or user).
* **Role-Based Data Retrieval**: The system checks the user's role when fetching patient data. The ‘retrieve\_patient\_data’ function displays different data based on the role, showing all fields for admins and limited fields for users.

**A screenshot of a computer

Description automatically generated**

The above screenshot shows User registration and the role-based access of AdminH, and options to retrieve and access details as per the admin role.

**Effectiveness:** Ensures that only authorized users can access or modify sensitive data, adhering to the principle of least privilege.

**Testing Scenario**:

* **Register Two Users**: Create one admin and one user account.
* **Data Retrieval**: Log in as an admin to view all patient data fields, then log in as a user and verify that first name and last name are not visible.

A white background with black text

Description automatically generated

A white background with black text

Description automatically generatedA computer screen shot of a computer code

Description automatically generatedA screenshot of a computer

Description automatically generated

The above screenshot shows the respective role assignment and data visible to users as per the role based data retrieval access.

**3. Basic Query Integrity Protection Requirements**: Detect modified data and incomplete query results.

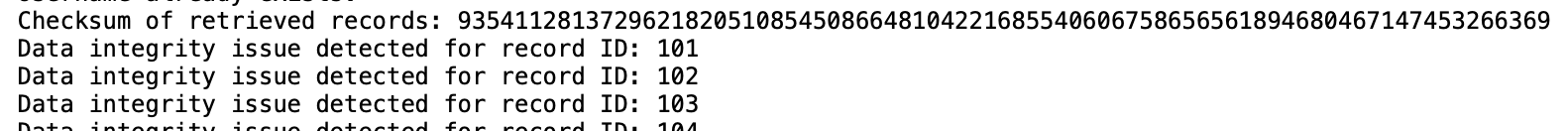
**Implementation**: SHA-256 hashes are generated for patient records to check data integrity.

* **Data Integrity Checks**: Each patient record includes a ‘record\_hash’ (SHA-256 hash). This hash is compared against a recalculated hash of the record's data to detect modifications. The system generates a SHA-256 hash of each patient record (‘generate\_record\_hash’ function) before storing it in the database. This hash can be used to verify the integrity of the data.
* **Incomplete Results Detection**: This is implemented by using a form of aggregate checksum or hash for a set of records, which can be compared with the client-side calculation. The checksum value is a computed number representing a form of the aggregate digital signature of all the records retrieved by the query. Its primary purpose here is to quickly indicate if any records have been added or removed from the dataset as a whole since the checksum would change if the overall content of the dataset changes.

**Effectiveness:** Helps in detecting unauthorized alterations of data, ensuring data integrity.

**Testing Scenario**:

* **Manual Record Modification**: Alter a patient's data in the database and update the ‘record\_hash’ accordingly.
* **Data Retrieval and Integrity Check**: Fetch data as a user or admin and observe if the system flags the modified record.



The above screenshot shows the check for Data Integrity and incomplete result detection checks.

**4**. **Basic Data Confidentiality Protection** **Requirements**: Secure sensitive attributes (gender and age) in the cloud, preventing statistical information leakage.

**Implementation**: Sensitive data (like gender and age) are encrypted using AES encryption before being stored in the database. Random padding is added to sensitive data before encryption.

* **Encryption of Sensitive Data**: The system encrypts gender and age data using AES encryption before storing them in the database, ensuring that these attributes are not stored in plaintext.
* **No Statistical Leakage**: The encryption method, combined with the padding strategy, helps prevent leakage of statistical patterns from the encrypted data.

**Effectiveness:** Protects sensitive data from being exposed in case of a database breach.Adds an additional layer of security, making it more difficult to infer information about the encrypted data.

**Testing Scenario**:

* **Database Inspection**: Verify that gender and age are stored in an encrypted form in the database.
* **Query Execution**: Perform queries as both an admin and a user, checking that decrypted values are correctly returned without revealing any discernible patterns or statistics directly from the database.

A screenshot of a computer code

Description automatically generated

The above screenshot shows the check for Data Integrity and Data Confidentiality checks to the implementation.

**3. Discussion of Limitations and Potential Vulnerabilities and Improvements (3 pts):**

* **Security and privacy-related limitations**

**-**The system pads and encrypts specific fields (gender, age) but leaves others (e.g., health history) potentially unencrypted, which might contain sensitive information.

**-** The implementation reveals detailed error messages, which might inadvertently expose system information to malicious users, hence the approach has scope of improvisation.

**-** The randomness used in data padding could be improved to ensure that the patterns are not predictable.

**-**The system generates AES keys but the approach to manage and store the keys securely can be improvised, as it is crucial for maintaining the confidentiality of encrypted data.

* **Potential Vulnerabilities and Improvements**

**-** Implement a secure key management system to handle encryption keys, possibly using hardware security modules (HSMs).

**-** Encrypt all sensitive patient data, not just selected fields.

**-** Refine error messages to avoid leaking system details while still being informative for legitimate debugging.

**-** Ensure that connections to the database are secure and encrypted, possibly using SSL/TLS.

**-** Regularly update the system and its dependencies to patch known vulnerabilities.

**4.Testing and Validation (Output Analysis):**

The testing and validation process, as demonstrated in the output, appears thorough and addresses key aspects of the system:

**Data Integrity:** Successfully detects tampering with patient data.

**Role-Based Access** **Control:** Validates that different user roles have appropriate access levels.

**User Authentication:** Confirms the effectiveness of the user authentication process.

**AES Encryption:** Validates the encryption and decryption of sensitive attributes.

**General Functionality:** Verifies the basic functionality of adding and retrieving patient data.

A close-up of a computer error

Description automatically generated

Overall, the architecture effectively integrates various security measures into a healthcare data management system, focusing on user authentication, data encryption, role-based access control, and data integrity. However, improvements in encryption key management, comprehensive data encryption, and error handling could further enhance its security and privacy capabilities.

**5.Team Members' Contribution**

This is an individual team member project and has been worked through and through by me (Priyanka Vyas). The file name enclosed to the submission as source code:

Enclosed Source code file: AES Encryption\_source\_code.ipynb and;

Gitlink: <https://github.com/vpriyanca/Data-Security-Project>

Reference:

*[1] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11). Association for Computing Machinery, New York, NY, USA, 85–100. https://doi.org/10.1145/2043556.2043566*