A blue and yellow logo

Description automatically generated

***Data Security and Privacy***

***Team members:***

***Chandan Kondapuram (811299920)***

***Pratik Airy (811310290)***

***Mehfoos Alam (811)***

**Introduction**

This project aims to develop a secure database system that ensures data confidentiality, integrity, and access control for multiple user groups. The system supports two user groups:

• Group H: Full access to all fields in the database and permission to add new data.

• Group R: Restricted access (cannot view sensitive fields like first\_name and last\_name) and no permission to add data.

**The system employs strong security measures:**

• User authentication using hashed passwords.

• Role-based access control.

• Query integrity checks to detect tampered or missing data.

• Encryption of sensitive fields (gender, age) to maintain confidentiality.

**Technologies Used:**

• Flask: Backend framework for API development.

• MySQL: Database management system.

• Python Libraries:

• bcrypt for hashing passwords.

• Fernet for encrypting sensitive fields.

**Background and Objectives**

Conventional database services often present a dilemma for users: they must choose between the strong security of local systems and the flexibility offered by cloud-based solutions. While encrypting data significantly enhances its protection, it can also introduce complexities in operations such as searching and retrieval.

This project seeks to address these ongoing challenges with a comprehensive approach that includes:

- Implementing robust user authentication mechanisms: We will develop advanced methods for verifying user identities, ensuring that only authorized individuals can access sensitive data.

- Establishing role-based access controls: By creating distinct access levels tailored to the needs of various user groups, we aim to protect sensitive information while allowing appropriate access to those who require it for their tasks.

- Protecting query integrity: Our system will incorporate measures that detect any tampering or incomplete results, safeguarding the accuracy and reliability of data retrieval.

- Ensuring data confidentiality: Special focus will be placed on protecting sensitive attributes, ensuring that private information remains secured from unauthorized access.

Our overarching objective is to create an innovative system that empowers users to leverage the advantages of cloud services, all while upholding rigorous standards of security and maintaining data integrity at every level.

**System Design**

**The architecture consists of three main components:**

1. Users (Group H and Group R):

• Group H: Full access to all database fields, permission to add data.

• Group R: Restricted access and no permission to add data.

1. Flask Application:

• Handles authentication, access control, encryption/decryption, and query integrity checks.

1. MySQL Database:

• Stores:

• Encrypted sensitive fields (gender, age).

• Row hashes for tamper detection.

• User details with hashed passwords.

**Architecture Diagram**

User

(Group H / Group R)

Requests: Registration, Login, Query, Add Data

Flask Application

* Authentication
* Encryption/Decryption
* Access Control
* Query Integrity Checks

Data Storage: Encrypted Fields, Row Hashes

MySQL Database

* Encrypted Fields
* Row Hashes for Integrity
* Hashed Passwords

**Security Features**

**a. User Authentication**

**Implementation**:

• Users register with a username, password, and group.

• Passwords are hashed using bcrypt before being stored in the database.

• During login, the hashed password is validated against the stored hash.

**Security**:

• Storing hashed passwords ensures the database never contains plaintext credentials, making it resilient to breaches.

**b. Access Control**

**Implementation**:

• Group H:

• Full access to all fields.

• Permission to add new records via the /add endpoint.

• Group R:

• Restricted access (cannot view first\_name or last\_name).

• No permission to add records.

**Security**:

• Role-based restrictions prevent unauthorized access and maintain data confidentiality.

**c. Query Integrity**

**Implementation**:

• Each record includes a row\_hash, a hash of its content.

• An overall\_hash ensures query completeness by hashing all returned rows.

**Security**:

• Any tampering with individual records or missing records results in hash mismatches, ensuring data integrity.

**d. Data Confidentiality**

**Implementation**:

• Sensitive fields (gender, age) are encrypted using Fernet before storage.

• Fields are decrypted during query retrieval for authorized users (Group H).

**Security**:

• Encryption ensures that sensitive fields remain confidential even if the database is compromised.

**Data Flow**

**Step 1: User Registration**

• Users register via the /register endpoint.

• Passwords are hashed using bcrypt and stored in the database.

**Step 2: User Login**

• Users log in via the /login endpoint.

• Credentials are validated against hashed passwords.

**Step 3: Query Execution**

• Users query data via the /query endpoint:

• Group H: Full access to all fields.

• Group R: Restricted access.

• Flask verifies integrity using row\_hash and overall\_hash.

**Step 4: Adding Data**

• Group H users add data via the /add endpoint.

• Sensitive fields are encrypted, and a row\_hash is calculated for each record.

**Team Contributions**

**Testing Results**

**a. User Authentication**

• **Success**: Users can register and log in with valid credentials.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer

Description automatically generated

A computer screen with white text

Description automatically generated

• **Failure Scenarios**:

• Invalid username/password returns an error.

A screen shot of a computer

Description automatically generated

**b. Access Control**

• Group H: Access to all fields and ability to add data.

A screenshot of a computer

Description automatically generated

• Group R: Restricted access (no first\_name or last\_name) and no add permissions.

A screenshot of a computer

Description automatically generated

**c. Query Integrity**

• Detecting Modified Data (Single Data Item Intergrity)

Manually modify a database entry (e.g., change weight).

A screenshot of a computer

Description automatically generated

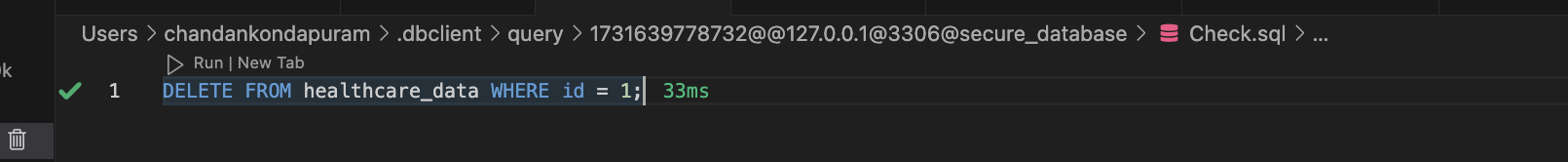
Query the data:

A screenshot of a computer

Description automatically generated

• Removed Data Item from a query (Query Completeness)

Removed Data Manually:



Result:

A screenshot of a computer

Description automatically generated

**d. Data Confidentiality**

• Encrypted fields (gender, age) are only decrypted for authorized queries.

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Limitations**

**a. Encryption Key Management**

• The encryption key is stored locally (cipher.key), making it a single point of failure.

**b. Query Completeness**

• Attackers could manipulate the overall\_hash to bypass missing record detection.

**c. Performance Overhead**

• Encryption and hash calculations may cause slow performance as data scales.

**d. Weak Password Policy**

• The system does not enforce strong password rules, making accounts vulnerable.

**Conclusion**

This project successfully implemented a secure database system with features like user authentication, role-based access control, query integrity checks, and data confidentiality. While the system meets all the project requirements, addressing the limitations (e.g., key management and password policies) would further enhance its security