Secure Hospital Management System (Kivy/Python)

This document provides an updated overview of the **Hospital Management System (HMS)**, a standalone desktop application built using the Kivy framework in Python. The application is designed with a strong emphasis on **practical security implementation**, where deployed controls are applied directly to live system components such as the actual Login Form and Patient Record modules.

---

1. Architecture Overview

The application is fully self-contained within the `main.py` file, incorporating both the front-end (Kivy UI) and back-end (Python logic, SQLite database management, and encryption).

Key Architectural Components

Component	Technology / Concept	Purpose
User Interface	Kivy (Python UI Framework)	Provides a responsive, cross-platform graphical interface for users.
Data Security	Fernet (AES Symmetric Encryption)	Encrypts sensitive patient data at rest within the database.
Persistent Storage	SQLite (Embedded Database)	Uses two separate database files for isolation and integrity.

Audit Logging	Dedicated Audit Database	Immutable record of all key user and system actions (logins, deletions, record changes).
Security Assurance	Parameterized SQL Queries	Prevents SQL Injection by sanitizing all user inputs in live system forms.

✅ All security controls are now applied directly to the **actual operational modules** (Login and Patient Records), not separate test modules.

---

2. Security Features

A. Data Isolation and Integrity

The application implements a **Dual Database Architecture** to ensure separation of concerns and resilience against compromise:

- **hms_patient_data.db**
  - Stores sensitive patient records
  - Personally identifiable fields (Name, Medical Condition) are encrypted using Fernet/AES
- **hms_audit_log.db**
  - Stores system activity logs
  - Records login attempts, record creation, updates, and deletions
  - Maintained separately to preserve log integrity even if patient database is compromised

Additionally, the **users table** is stored in the patient database and is used for real authentication via SQL queries.

B. Encryption (Fernet / AES)

A system-generated key stored in `hms_key.key` is used by the Fernet library to encrypt and decrypt sensitive patient data. Data is:

- Encrypted before storage
- Decrypted only temporarily when viewed on the dashboard
- Never stored in plaintext within the database

This ensures confidentiality even if the database file is accessed externally.

C. SQL Injection Prevention (Live Implementation)

The system actively implements **SQL Parameterization** on all critical forms:

✅ Login Form
✅ Add User Form
✅ Patient Record Entry Form

Example secure query pattern used:

1. cursor.execute(
2.   "SELECT * FROM users WHERE username = ? AND password = ?",
3.   (username, password)
4.)

Because of this, attempts to inject commands such as:

5. ' OR '1'='1

will fail and be logged as `FAILED_CREDS`, proving the effectiveness of the control.

This demonstrates that SQL Injection protection is actively deployed and validated within the actual system workflow, fulfilling the intended security objective.

3. Getting Started

A. Prerequisites

- Python 3.x
- Kivy Framework
- cryptography library (for Fernet encryption)

B. Installation

Install required dependencies:

6. pip install kivy
7. pip install cryptography

C. Running the Application

Execute the application using:

8. python main.py

D. Default Credentials

The application starts on the Login Screen.

Field	Value
Username	admin
Password	admin123

Additional users can be added via the **Add User GUI screen**, which also uses parameterized SQL for security.

---

4. Evidence of Security Control Deployment

The following confirm successful implementation:

- Login attempts using `'` or injection strings fail
- Audit log records show SUCCESS and FAILED_CREDS statuses
- Screenshot evidence includes:
    - Original login attempt using SQL injection input
    - Resulting failed login
    - Corresponding audit log entry

These screenshots prove that SQL parameterization is not theoretical but enforced in the system.

Summary of Improvements

✔ Security controls now applied to actual system forms
✔ No separate testing module is used
✔ Live SQL Injection prevention demonstrated
✔ Dual database architecture preserved
✔ Audit logging fully functional

This updated implementation now aligns with the system objectives of testing deployed security controls on real operational components, fulfilling the instructor's feedback requirements.