# Encrypted File Storage System with Custom Cipher

**Project Overview**

The **Encrypted File Storage System with Custom Cipher** is a secure file storage solution that leverages a custom encryption algorithm to protect sensitive files. The system is designed to encrypt files using a novel encryption method, store the encryption key securely, and allow for decryption when required. The project includes a database system for managing encryption keys and tracking file access logs, providing a secure, efficient way to manage and protect files.

---

**File Structure and Data Flow**

**1. main.py**

- **Role:** This is the entry point of the project. It presents the user with options to either encrypt or decrypt a file. Based on user input, it coordinates the flow of control to other components like file encryption, key management, and logging.
- **Functionality:**
  - Prompts the user to choose between encryption or decryption.
  - Collects user input for file name and ensures that the file exists.
  - Calls functions from other files to perform encryption or decryption.
- **Data Flow:**
  - User input (file name and action) is collected.
  - Based on the action, the system retrieves the corresponding encryption key from the database and performs the requested action on the file.
  - The access logs are updated in the database.

**2. encryption.py**

- **Role:** This file handles the custom encryption and decryption logic. The encryption algorithm transforms the file content based on a custom cipher algorithm.
- **Functionality:**
  - Encrypts the file content using the provided key and the custom cipher.
  - Decrypts the file content back to its original form using the same key.
- **Data Flow:**
  - The file content is read, transformed using the custom cipher, and written back to the file.
  - During encryption, the encrypted content is saved to the same file, overwriting the original content.

**3. database.py**

- **Role:** This file handles the SQLite database operations, including creating tables, saving file encryption keys, and logging file access.

- **Functionality:**
    - Initializes the SQLite database with two tables: one for storing file encryption keys and another for logging access events.
    - Stores and retrieves encryption keys for each file.
    - Logs access events (e.g., encryption or decryption) along with a timestamp.
- **Data Flow:**
    - Stores the encryption key generated for each file when it is encrypted.
    - Retrieves the correct encryption key when a file is decrypted.
    - Logs each encryption and decryption event for auditing and monitoring.

**4. insert_dummy_data.py**

- **Role:** This optional script allows the insertion of dummy data into the database for testing purposes.
- **Functionality:**
    - Inserts dummy records into the database tables to simulate file encryption and access logs.
- **Data Flow:**
    - Adds sample data into the files and access_logs tables for testing the functionality of the system.

---

**Features**

1. **Custom Encryption Algorithm**:
    - Implements a novel cipher for encrypting file content. This ensures data security by using a unique, user-defined encryption method.
2. **File Encryption and Decryption**:
    - Users can choose to encrypt or decrypt any file in the system. The encryption and decryption process is fully automated and integrates with the database for key management.
3. **Key Management**:
    - Encryption keys are securely stored in a SQLite database, linked to the corresponding file. The system retrieves these keys during decryption operations.
4. **Access Logging**:
    - Logs every file encryption or decryption operation, storing the file name, action performed, and timestamp. This provides an audit trail for security monitoring.

---

**Challenges Faced**

1. **Custom Cipher Design**:

- Designing a robust and efficient custom encryption algorithm required careful consideration of security principles. The encryption method needed to ensure that the data was sufficiently scrambled to protect against unauthorized access.

2. **File Handling and Overwriting**:
   - Managing file content during encryption and decryption was tricky, especially when overwriting existing files. Ensuring that the correct content was encrypted and written back without data loss was critical.
3. **Database Integration**:
   - Integrating a database to manage encryption keys and logs required careful design to ensure seamless interactions between the encryption system and the database.
4. **Security Considerations**:
   - While the custom cipher adds uniqueness to the project, ensuring that the encryption is secure enough for real-world applications remains a challenge. Custom algorithms need extensive testing to avoid vulnerabilities.

---

**Future Possibilities**

1. **Improved Encryption Algorithm**:
   - The custom cipher can be further enhanced for greater security. Future iterations can include more advanced cryptographic techniques like symmetric or asymmetric key encryption (e.g., AES, RSA).
2. **User Authentication**:
   - The system could be extended to include user authentication, ensuring that only authorized users can encrypt, decrypt, or access sensitive files.
3. **File Integrity Checks**:
   - Implementing file integrity checks using hash functions (e.g., SHA-256) could ensure that encrypted files have not been tampered with or corrupted.
4. **Cloud Integration**:
   - Future versions could integrate with cloud storage platforms, allowing users to encrypt files locally and store them securely in the cloud.
5. **Advanced Access Logs**:
   - Access logs could be enhanced with additional metadata like the user's IP address, location, and device details. This would provide deeper insights into access patterns and help detect suspicious activity.
6. **GUI Development**:
   - A graphical user interface (GUI) could be developed to make the system more user-friendly, allowing users to drag and drop files for encryption or decryption.

---

**Conclusion**

The **Encrypted File Storage System with Custom Cipher** provides a solid foundation for securely storing files with custom encryption. While the current system focuses on the core functionality of file encryption, key management, and logging, there are many opportunities to extend its capabilities, making it more secure and user-friendly.

The project illustrates key principles in cybersecurity and cryptography, and it can be a useful tool for learning and experimentation in the field of secure file storage.