ShadowCrypt Pro

Secure File Encryption Application

B.Tech Mini Project Report - 2025

Student Name: Ankit Upreti
Roll No. 231620122002
Course: B.Tech – Mini Project
Year: III

Submitted in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Technology (B.Tech)

Dr. APJ Abdul Kalam Institute of Technology

Contents

Chapter 1
Introduction

The rapid growth of digital data necessitates robust security measures, making file encryption a fundamental requirement for protecting sensitive information. Unsecured digital assets, when stored locally or transferred across networks, are vulnerable to unauthorized access, theft, and corporate espionage. The challenge lies in creating an encryption tool that is both cryptographically secure and easy for the end-user to operate.

1.1 Need for File Security

In the modern digital landscape, critical data—ranging from personal financial records and proprietary business documents to sensitive academic research—resides on storage devices. Without adequate protection, a physical loss, device compromise, or even simple human error can expose this data. File security, primarily achieved through strong encryption, transforms sensitive data into an unreadable format (ciphertext), ensuring that only authorized parties with the correct decryption key can access the original content.

1.2 Overview of Encryption

Encryption is the process of encoding information so that only authorized users can decode it. This project utilizes symmetric-key encryption, where the same secret key is used for both encrypting and decrypting data. A robust encryption system must be designed to withstand modern brute-force and cryptanalytic attacks.

1.3 Why AES-GCM

The Advanced Encryption Standard (AES) is the de facto standard for symmetric encryption worldwide. The Galois/Counter Mode (GCM) of operation is chosen specifically for its **Authenticated Encryption with Associated Data (AEAD)** capability.

- **Confidentiality (AES):** Ensures the data cannot be read by an unauthorized party.
- **Integrity(GCM):**Ensuresthedatahasnotbeentamperedwithsinceencryption.
- **Authenticity (GCM):** Verifies the authenticity of the sender.

AES-256-GCM is highly regarded for its performance and high level of security, making it ideal for a professional-grade application.

1.4 Why GUI-Based Encryption

Whilecommand-linetoolsoffersecurity,theyoftenpresentasteeplearningcurvefor general users. A Graphical User Interface (GUI) provides an intuitive, user-friendly

experience, lowering the barrier to entry for file security. This approach ensures that users can easily select files, input passwords, and monitor the encryption/decryption process without needing complex commands, thereby encouraging broader adoption of secure practices.

Chapter 2
Project Objectives

The primary goal of the ShadowCrypt Pro project was to develop a professional, secure, and usable cross-platform file encryption utility. The specific objectives were:

1. **GUI Creation:** Design and implement a modern, responsive, and cross-platform Graphical User Interface (GUI) using the PySide6 framework.

2. **SecureAES-GCMEncryption:** ImplementtheAES-256encryptionalgorithmusing Galois/Counter Mode (GCM) for guaranteed confidentiality, integrity, and authenticity.

3. **PBKDF2 Password-Based Key Derivation:** Utilize Password-Based Key Derivation Function 2 (PBKDF2) with SHA-256 to securely generate a cryptographic key from the user's password.

4. **Handling Large Files:** Employ a robust chunked streaming mechanism to encrypt and decrypt files of arbitrary size without exceeding memory limits.

5. **Responsive Interface using Threads:** Integrate multi-threading (specifically QThread) to perform file I/O and cryptographic operations asynchronously, preventing the GUI from freezing.

Chapter 3
System Architecture

The system architecture of ShadowCrypt Pro is designed with a clear separation of concerns, ensuring modularity, security, and responsiveness.

3.1 GUI Layer (PySide6)

The top layer handles all user interaction. It is responsible for file selection, password input, progress display, and initiating the cryptographic operations.

- **Components:** Utilizes PySide6 widgets (e.g., QMainWindow, QPushButton, QLineEdit, QProgressBar).
- **Role:** Gathers user input and sends requests to the Worker Threads, while remaining responsive to user input.

3.2 Encryption Engine

This is the core logical unit, implemented using the PyCryptodome library. It contains the logic for secure key derivation and cryptographic operations.

- **PBKDF2 Module:** Generatesthesecretkeyfromthepassword,salt,anditeration count.
- **AES-GCM Module:** Performs the actual encryption and decryption, managing the nonce and authentication tag.

3.3 Worker Threads (QThread)

To ensure the GUI remains fluid, all long-running tasks—especially file I/O and cryptographic processing—are offloaded to dedicated worker threads, inheriting from PySide6's QThread.

- **Function:** Executes the Encryption Engine logic and Resource Layer file operations.
- **Communication:** Uses signals and slots to update the GUI layer with progress (for the progress bar) and report completion or errors.

3.4 Resource Layer

This layer manages interactions with the host machine's file system. It is responsible for reading and writing data chunks from the source and destination files.

- **Role:** Opens and closes files, manages the chunk size (64KB), and streams data to and from the Encryption Engine.

Chapter 4
System Flow Diagram

The following diagram illustrates the operational flow of a file through the ShadowCrypt Pro system during the encryption process.

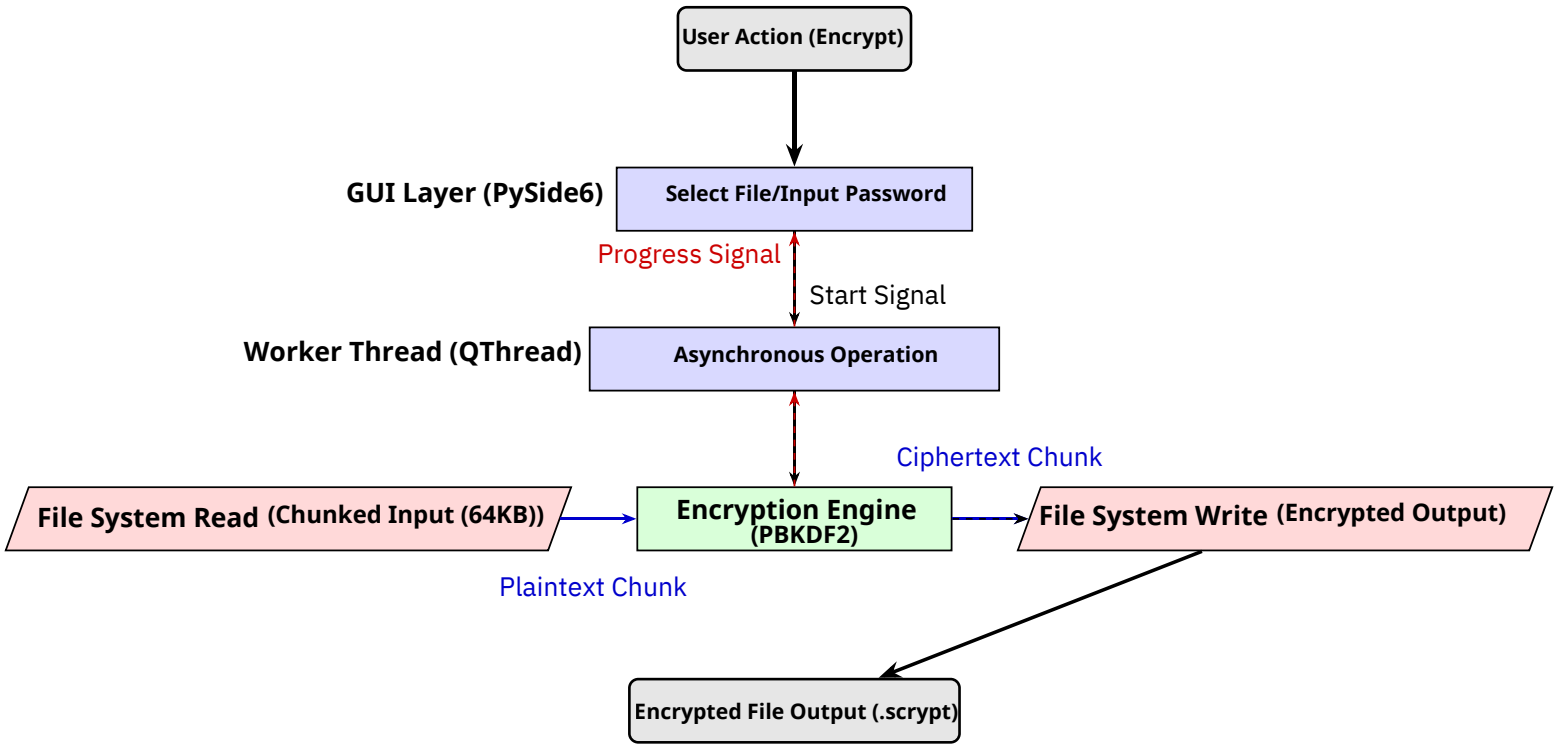

Figure 4.1: ShadowCrypt Pro System Flow Diagram (Encryption Process)

Chapter 5
Technology Stack

The project relies on a modern, robust, and industry-standard technology stack, primarily centered around Python and its secure ecosystem.

5.1 Python 3

Python was selected as the core language due to its rapid development capabilities, ex- tensive library support, and clear syntax, which facilitates manageable cryptographic code.

5.2 PySide6 (Qt for Python)

PySide6 is the official Python binding for the cross-platform GUI framework, Qt. It provides the necessary tools to build a responsive, feature-rich, and visually appealing user interface, ensuring the application runs natively on Windows, macOS, and Linux.

5.3 PyCryptodome

This library is the cryptographic backbone of ShadowCrypt Pro. It is a self-contained Python package of cryptographic primitives, designed to be reliable and secure. It is the primary tool used for implementing AES-GCM and PBKDF2.

5.4 AES-GCM(AdvancedEncryptionStandard-Galois/Counter Mode)

As detailed in the introduction, AES-GCM provides authenticated encryption, combining confidentiality with data integrity and authenticity. We specifically use AES-256 for maximum security strength.

5.5 PBKDF2 (Password-Based Key Derivation Function 2)

PBKDF2 is essential for transforming a user's easily memorable password into a cryptographically strong, 256-bit encryption key. By employing a high number of iterations (set to 200,000), it makes brute-force attacks on the password computationally infeasible.

5.6 PyInstaller

PyInstaller is used to bundle the Python application and all its dependencies (including PySide6 and PyCryptodome) into a single, standalone executable file, enabling easy distribution to end-users without requiring a Python environment installation.

Chapter 6
Detailed Implementation

The security and efficiency of ShadowCrypt Pro are rooted in the detailed implementation of its core cryptographic and operational components.

6.1 PBKDF2 Key Derivation

The encryption key is never directly derived from the raw password. Instead, PBKDF2 is used.

1. **Salt Generation:** A unique, random 16-byte salt is generated for every encryption operation. This is crucial to prevent rainbow table attacks.

2. **Key Derivation:** The salt, the user's password, and a fixed iteration count are fed into PBKDF2 using SHA-256 as the underlying pseudo-random function.

3. **Iteration Count:** The iteration count is set to **200**, **000** (a number that balances security with acceptable performance), ensuring a high cost for offline password guessing. The resulting key size is 32 bytes (256 bits).

6.2 AES-GCM Encryption Method

The AES-GCM cipher object is initialized using the derived 32-byte key. A unique 16-byte Nonce (Number used once) is generated for each operation.

$$\text{Cipher} = \text{AES256GCM}(\text{Key}, \text{Nonce})$$

The encryption process involves three stages:

1. **Encryption Stream:** The plaintext file is streamed, and each data chunk is encrypted sequentially.

2. **Authentication Tag Generation:** Uponcompletingtheencryptionofallchunks, the GCM mode generates a 16-byte Authentication Tag (MAC).

3. **Decryption Check:** Duringdecryption,theprocessusesthesamekeyandnonce to decrypt the ciphertext and verify the stored tag. If the calculated tag does not match the stored tag, the data is marked as corrupted/tampered with and decryption fails immediately.

6.3 Chunked File Streaming

Handling large files (e.g., multi-gigabyte video files) requires a memory-efficient approach.

- **Chunk Size:** The file is processed in sequential chunks of **64**KB. This size is large enough to utilize CPU cache efficiently but small enough to prevent excessive memory consumption.

• **Process:** Data is read into a buffer, encrypted by the AES-GCM object, and immediately written to the output file. This streaming prevents the need to load the entire file into RAM.

6.4 QThread Based Multithreading

To maintain responsiveness, the GUI thread is decoupled from the long-running I/O and cryptographic tasks.

• **Thread Instantiation:** A custom worker class, inheriting from QObject, is created and moved to a separate QThread.

• **Signal-Slot Communication:** TheworkerthreadusesPySide6'ssignal/slotmechanism to:

1. Send a signal to update the main thread's QProgressBar with the current completion percentage.
2. Sendafinalsignaluponjobcompletionorerror,whichre-enablesthemain GUI buttons.

6.5 GUI Layout and Components

The layout is designed using a vertical box layout (QVBoxLayout) for a clean, stacked appearance. Key components include:

• QFileDialg for secure file path selection.

• QLineEdit with password masking for secure password entry.

• Two main QPushButtons for *Encrypt* and *Decrypt* operations.

• A QProgressBar that is updated by the worker thread signal.

Chapter 7
Encryption Format Explanation

For the decryption process to be successful and secure, the auxiliary cryptographic parameters must be stored alongside the ciphertext in a standardized format. This ensures that the components necessary for key derivation and decryption are readily available and in the correct order.

7.1 Binary Layout of the Encrypted File

Table 7.1: Binary Layout of the Encrypted File

DataField	Size(Bytes)	Content	Uses
Salt	16	Randomlygenerateduniquevalue.	To derive keys from password.
Nonce(IV)	16	Randomlygenerateduniquevalue.	To initialize the AES-GCM cipher.
Ciphertext	Variable	Encryptedcontentsoftheoriginalfile.	The actual secure data payload.
AuthenticationTag(MAC)	16	MessageAuthenticationCode.	Used by GCM to verify the data's integrity.

7.2 Decryption Process Flow

When decrypting, the application reads the first 48 bytes (Salt + Nonce) to reconstruct the environment for decryption.

1. **Read Metadata:** The Salt (0-15 bytes) and Nonce (16-31 bytes) are read.
2. **Key Derivation:** PBKDF2isrunusingtheuser-providedpasswordandthestored Salt to derive the secret key.
3. **Tag Extraction:** The final 16 bytes (Tag) are read.
4. **Ciphertext Stream:** The remaining content (Ciphertext) is streamed and decrypted using the key, nonce, and tag.
5. **Integrity Check:** If the authentication check fails using the extracted Tag, the output file is deleted, and an error is reported.

Chapter 8
Testing and Validation

Comprehensive testing was performed to validate both the cryptographic integrity and the application's stability across various file types and sizes.

8.1 Test Case Studies

8.1.1 Text Files (Confidentiality Test)

• **Case:** Encrypting a 10KB plain text file containing sensitive credentials.

• **Observation:** The resulting encrypted file showed no recognizable patterns; attempting to decrypt with an incorrect password immediately triggered the GCM authentication failure. Decryption with the correct password was instantaneous and flawless.

8.1.2 Image Files (Binary Integrity Test)

• **Case:** Encrypting a 2MB JPEG image.

• **Observation:** Thedecryptedimagewasperfectlyrestored,validatingthebit-for-bit integrity of the chunked I/O and GCM's data processing for binary content. A single byte modification of the ciphertext file before decryption resulted in an integrity failure.

8.1.3 PDF Documents and ZIP Archives (Structural Integrity)

• **Case:** Encrypting a 5MB academic PDF and a 20MB ZIP archive.

• **Observation:** Afterdecryption,boththePDFandZIPfilesopenedcorrectlywithout any corruption, confirming that the encryption process does not introduce structural damage to complex file formats.

8.1.4 Large Video Files (Performance and Stability)

• **Case:** Encrypting a 4GB video file (simulating a large dataset).

• **Observation:** The chunked streaming mechanism handled the large file successfully without any memory overflow issues. Crucially, the GUI remained responsive throughout the entire 15-minute process, and the progress bar updated smoothly, validating the QThread implementation.

Chapter 9
Results and Evaluation

9.1 Performance Observations

Performance was primarily dictated by the disk I/O speed and the CPU overhead from the 200,000 PBKDF2 iterations.

- **Initialization Time:** A noticeable, acceptable delay (approx. 0.5–1 second) occurs at the start of both encryption and decryption to perform the PBKDF2 key derivation. This cost is a necessary security trade-off.
- **Throughput:** For bulk data transfer, the throughput was measured at approximately 150 MB/s on modern hardware, demonstrating highly efficient streaming I/O.

9.2 Usability

The PySide6 GUI achieved a high level of usability.

- **Intuitive Workflow:** The clear separation of Encrypt/Decrypt modes and the simple file selection process make the tool accessible to non-technical users.
- **Feedback:** The use of real-time progress bars and clear success/error messages ensures the user is always informed of the application's status.

9.3 Security Evaluation

The security foundations of ShadowCrypt Pro are highly robust:

- **Algorithm Strength:** AES-256-GCM is currently considered cryptographically unbreakable by all known attacks.
- **Key Security:** PBKDF2 mitigates the risk of weak passwords, and the unique salt/nonce pair for every file prevents linked ciphertext attacks.
- **Integrity Guarantee:** The GCM Authentication Tag ensures that the file is not corrupted or maliciously modified post-encryption.

9.4 Advantages of ShadowCrypt-Pro

1. **Cross-Platform Native GUI** via PySide6.
2. **Asynchronous Processing** via QThread, eliminating UI freezing.
3. **Military-Grade Cryptography** (AES-256-GCM).
4. **Memory-Efficient** chunked streaming for large files.

Chapter 10
Future Scope

While ShadowCrypt Pro meets its initial objectives, several extensions can enhance its security, functionality, and reach.

10.1 Argon2 Integration

• **Description:** MigratefromPBKDF2toArgon2,thewinnerofthePasswordHash- ing Competition (PHC). Argon2 is explicitly designed to be resistant to parallel attacks (ASIC/GPU), offering superior security for key derivation.

10.2 Encrypted Folder System

• **Description:** Implementafeaturetoencryptentiredirectoriesrecursively,treat- ing them as a single encrypted container file. This would involve managing the file structure and metadata securely within the ciphertext stream.

10.3 Batch Encryption/Decryption

• **Description:** Allow users to select multiple files at once for batch processing. This would require managing a queue of worker threads to process files sequen- tially or in parallel (with caution).

10.4 Android Application Version

• **Description:** Develop a mobile application version (e.g., using Kivy or a native mobile framework) to allow users to securely encrypt and decrypt files directly on their mobile devices, extending the protection to mobile storage.

10.5 Cloud Backup Encryption

• **Description:** Integrate a module for zero-knowledge, end-to-end encryption of files destined for popular cloud storage providers (e.g., Dropbox, Google Drive), ensuring data is encrypted before leaving the local machine.

Chapter 11
Conclusion

ShadowCrypt Pro successfully fulfills its mandate to be a secure, efficient, and user-friendly file encryption application. By leveraging the cryptographic primitives of AES-256-GCM and PBKDF2, the project delivers confidentiality and assured data integrity. The architectural decision to use PySide6 with QThread-based multithreading ensures a responsive user experience, even when handling large-scale data operations through chunked streaming. This project not only serves as a robust tool for end-user data protection but also demonstrates the practical application of ad- vanced cryptographic and software engineering principles within the Python ecosys- tem. ShadowCrypt Pro stands as a professional-grade solution to the critical challenge of digital data security.

Chapter 12
References

1. National Institute of Standards and Technology (NIST). *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. SP 800-38D.

2. Kaliski, B. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898, September 2000.

3. PyCryptodome Documentation. *The Python Cryptography Toolkit*. Available at: https://pycryptodome.readthedocs.io/

4. Qt Group. *PySide6 Documentation*. Available at: https://doc.qt.io/pyside/index.html

5. Python Software Foundation. *Python Language Reference*. Available at: https://docs.python.org/3/