An Efficient Desktop-Based File and Folder Encryption Framework Using Fernet Cryptography

Shruti Dalela

Research Scholar, India

DOI: https://doi.org/10.51244/IJRSI.2026.1305000288

Received: 21 May 2026; Accepted: 26 May 2026; Published: 17 June 2026

ABSTRACT

Data security has become one of the most critical concerns in the modern digital era due to the rising frequency of cyberattacks, unauthorized file access, ransomware infections, and data theft. Personal users, students, organizations, and businesses continuously store confidential information in digital format, making robust file protection essential. This research paper presents a desktop-based cybersecurity application named Secure File & Folder Encryptor, developed using Python 3.13.3, Tkinter GUI, and Fernet symmetric encryption from the Python Cryptography library. The application allows users to encrypt and decrypt individual files, folders, images, and ZIP archives securely on the Windows platform. A unique cryptographic key is generated per encryption operation and stored as a .key file, while the ciphertext is saved as a .enc file. The system implements AES-128-CBC with HMAC-SHA256 for authenticated encryption, ensuring confidentiality, data integrity, and tamper detection. The software is distributed as a Windows Setup installer (.exe) requiring no Python environment. Future enhancements include migration to a Django/Flask web platform with AI-based threat analysis and cloud storage integration.

Keywords: Cybersecurity, Fernet Cryptography, Symmetric Encryption, File Encryption, Folder Encryption, Tkinter, Python, AES-128-CBC, HMAC-SHA256, Data Security, Windows Application, Secure File System.

INTRODUCTION

In modern computing environments, digital data is highly vulnerable to unauthorized access, cyberattacks, malware, ransomware, and insider threats. Individuals and organizations often store important documents, images, financial records, project files, and confidential data on local systems without proper protection mechanisms. Data breaches, ransomware attacks, corporate espionage, and unauthorized access to sensitive documents are increasingly common threats that affect users at every level.

Traditional password protection mechanisms are not sufficient to guarantee data confidentiality because files can still be stolen, copied, or modified by attackers. Therefore, encryption-based cybersecurity systems are necessary to protect digital information. The need for accessible, reliable encryption tools—usable without deep technical expertise—has never been greater.

This research introduces a desktop-based encryption framework called Secure File & Folder Encryptor, developed using Python 3x (3.13.3) and Tkinter GUI technology. The application uses Fernet Cryptography to encrypt files and folders securely using a generated secret key. Only users possessing the correct .key file can decrypt the protected data. The application is specially designed for students, teachers, cybersecurity learners, office workers, businesses, government departments, personal computer users, researchers, and developers who require a lightweight, user-friendly local encryption solution.

The system supports file encryption and decryption, folder encryption and decryption, whole-folder secure encryption, image encryption, secure key generation, and Windows setup installation. The application source code is implemented in Python using Tkinter GUI and Fernet cryptographic algorithms, demonstrating the practical execution of these functionalities in a real desktop environment.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Problem Statement

Many users store sensitive files on local systems without any form of encryption protection. When a device is stolen, hacked, or infected by malware, confidential data becomes immediately accessible to attackers. The growing frequency of data breaches, ransomware infections, and insider threats has made local file security a critical concern for individuals and organizations alike.

Common problems faced by users include unauthorized file access, data theft, leakage of confidential documents, ransomware attacks, and the absence of beginner-friendly encryption tools. Existing enterprise-grade encryption solutions are often expensive, complicated, or require advanced technical knowledge that is beyond the reach of ordinary users. There is therefore a clear need for a lightweight, user-friendly, desktop-based encryption framework that provides strong cryptographic security while remaining simple enough for non-technical users.

Objectives Of the Proposed System

The main objectives of the proposed Secure File & Folder Encryptor application are as follows:

- To provide secure encryption and decryption of files and folders using key-based cryptography.
- To protect confidential digital data using industry-standard Fernet (AES-128-CBC + HMAC-SHA256) cryptography.
- To automatically generate a unique cryptographic key per encryption operation.
- To provide an easy-to-use graphical user interface (GUI) that requires no command-line or cryptographic expertise.
- To support multiple file types including images, ZIP folders, and documents with automatic plaintext deletion after encryption.
- To prevent unauthorized access to sensitive data by removing original files after successful encryption.
- To provide a Windows-based installable cybersecurity solution distributed as a Setup (.exe) installer.
- To lay the architectural groundwork for future migration to Django or Flask web-based platforms with cloud and AI integration.

Technology Stack & Cryptographic Architecture

The application employs a modern Python-based stack. Fernet—provided by the cryptography library (PyPI)—uses AES-128 in CBC mode for confidentiality with PKCS7 padding, a 128-bit IV generated freshly per encryption call, and HMAC-SHA256 for message authentication. The result is an Authenticated Encryption with Associated Data (AEAD) scheme protecting against both passive eavesdropping and active tampering. A unique key is generated via Fernet. Generate key (), yielding a URL-safe base64-encoded 256-bit key stored as a .key file.

Component	**Technology**
Programming Language	Python 3.x
GUI Framework	Tkinter (built-in)
Encryption Library	cryptography (PyPI) — Fernet, AES-128-CBC, HMAC-SHA256
Compression	Python zipfile — ZIP_DEFLATED

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

File I/O	Python os, shutil, tempfile modules
Target Platform	Windows (desktop .exe via Setup installer)
Future Stack	Django or Flask (web GUI, multi-user, browser-accessible)

Table 1: Technology Stack

System Architecture & Workflow

The application is composed of eight functional modules: User Interface Module, File Selection Module, Key Generation Module, Encryption Engine, Decryption Engine, Whole-Folder Encryption Module, Secure Key Validation Module, and File Restoration Module. The encryption and decryption workflows follow structured, deterministic pipelines as detailed below.

File Encryption Workflow

Step	**Operation**
1	User selects file via native Windows file dialog
2	Application generates a unique Fernet key and saves as <filename>.key
3	Original file bytes are read into memory
4	Original file extension is prepended to payload using delimiter |||
5	Combined payload is encrypted using Fernet, producing a ciphertext token
6	Ciphertext written to <filename>.enc
7	Original plaintext file is securely deleted (os. remove)

Table 2: File Encryption Workflow Steps

File Decryption Workflow

The user selects the .enc file and the corresponding .key file via sequential dialogs. The application validates the key using Fernet's token verification. If the key is correct, the original extension is recovered from the decrypted payload and the file is restored. The .enc file is then deleted. If an incorrect key is provided, Fernet raises an Invalid Token exception and a clear error dialog is displayed without revealing any information about the encrypted content.

Folder Encryption Workflow

Entire folder trees are encrypted in three stages: (1) the selected folder is compressed into a temporary ZIP archive using ZIP_DEFLATED compression, (2) a unique Fernet key is generated, and (3) the ZIP archive is encrypted using Fernet, producing a single .enc file. The original folder is removed after successful encryption. This approach reduces storage footprint while ensuring complete folder-tree protection.

Application Screenshots & Workflow

This section walks through the complete end-to-end usage of the Secure File & Folder Encryptor, illustrated with screenshots of every stage of the application interface on the Windows platform. The application features a dark-themed (#1e1e1e) main window with five clearly labelled action buttons and a status label displaying "Ready" when idle.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Main Application Window

When the application is launched, the main window appears with five clearly labelled action buttons: Encrypt File, Decrypt File, Encrypt Whole Folder, Decrypt Whole Folder, and Exit. The status label at the top displays "Ready", indicating the application is idle and awaiting user input.

Figure 1: Main Application Window

The Advanced File & Folder Encryptor main window showing Encrypt File, Decrypt File, Encrypt Whole Folder, Decrypt Whole Folder, and Exit buttons with status label displaying "Ready"

File Encryption Workflow

Clicking Encrypt File opens the native Windows file dialog, allowing the user to browse and select any file type — documents, images, spreadsheets, executables, and so on. After selection, a progress dialog immediately appears. On successful completion, a success message box displays the full paths of both the newly created .enc encrypted file and the .key key file. The original plaintext file is automatically deleted from disk at this point.

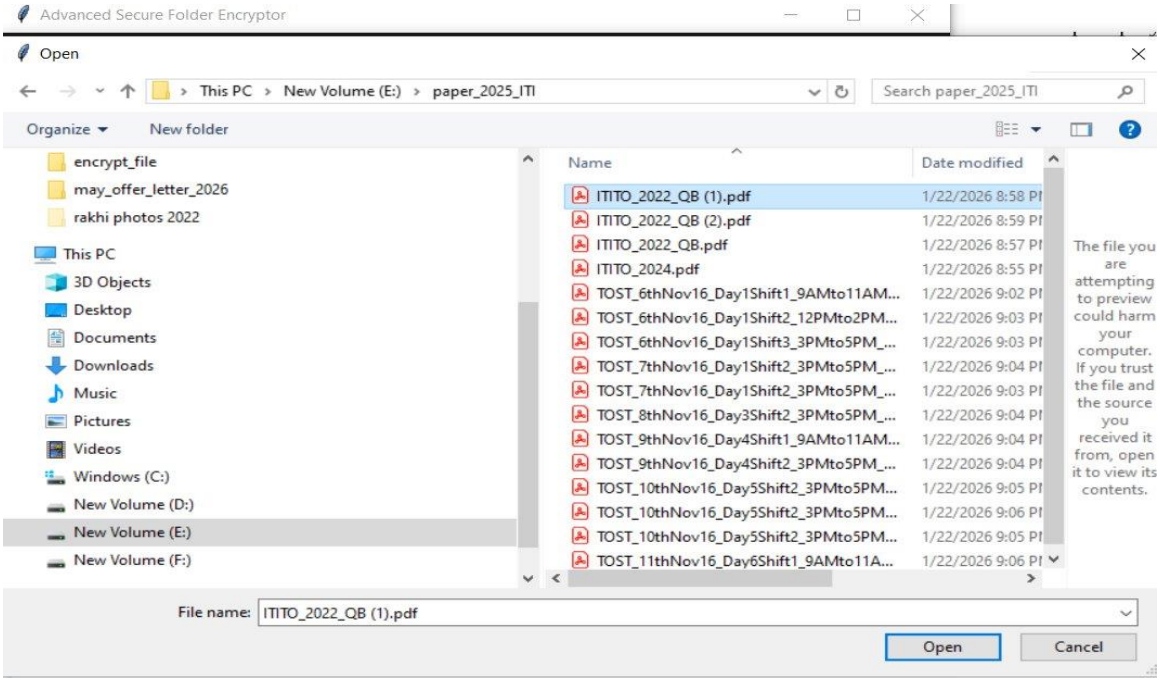

Figure 2: File Selection Dialog — Step 1 of Encryption

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Native Windows Open dialog for browsing and selecting a PDF file from the paper_2025_ITI folder to encrypt

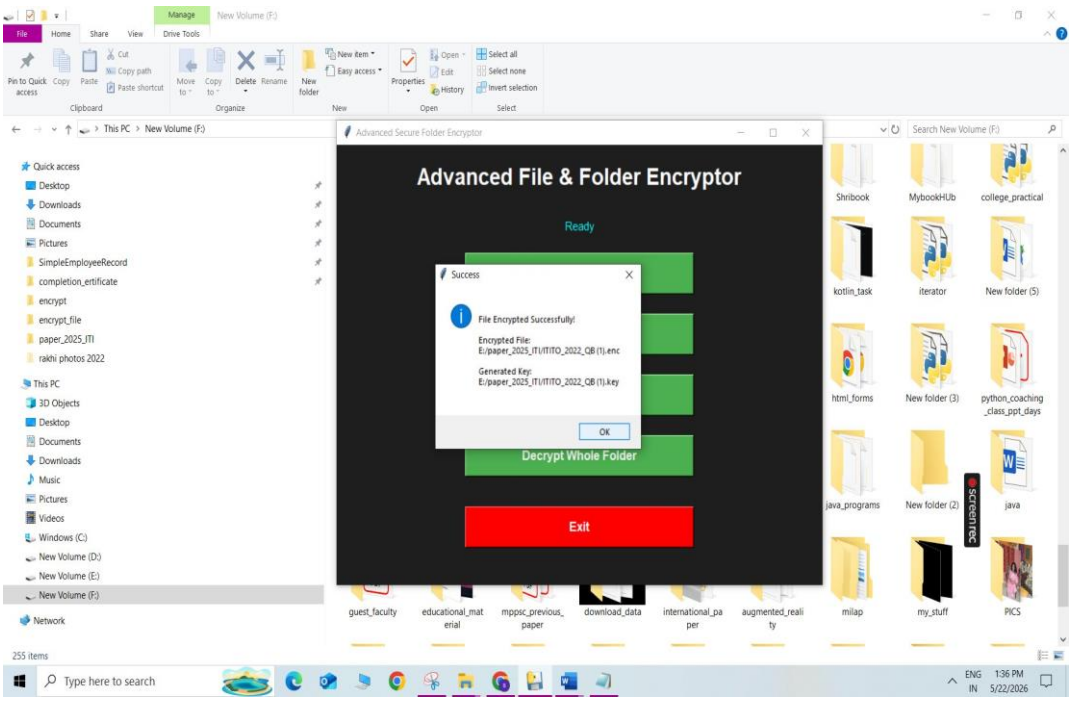

Figure 3: Encryption Success Dialog — Step 2 of Encryption

Success dialog confirming File Encrypted Successfully, showing the generated. enc encrypted file path and the .key key file path; original file has been deleted

File Decryption Workflow

Clicking Decrypt File opens two sequential file dialogs — first to select the. enc encrypted file, then to select the corresponding .key file. Both must be provided for decryption to succeed. Upon successful decryption, the application restores the file with its original extension and displays the restored file path. The .enc file is then automatically deleted. If the wrong .key file is provided, Fernet raises an Invalid Token exception and a clear error dialog is shown.

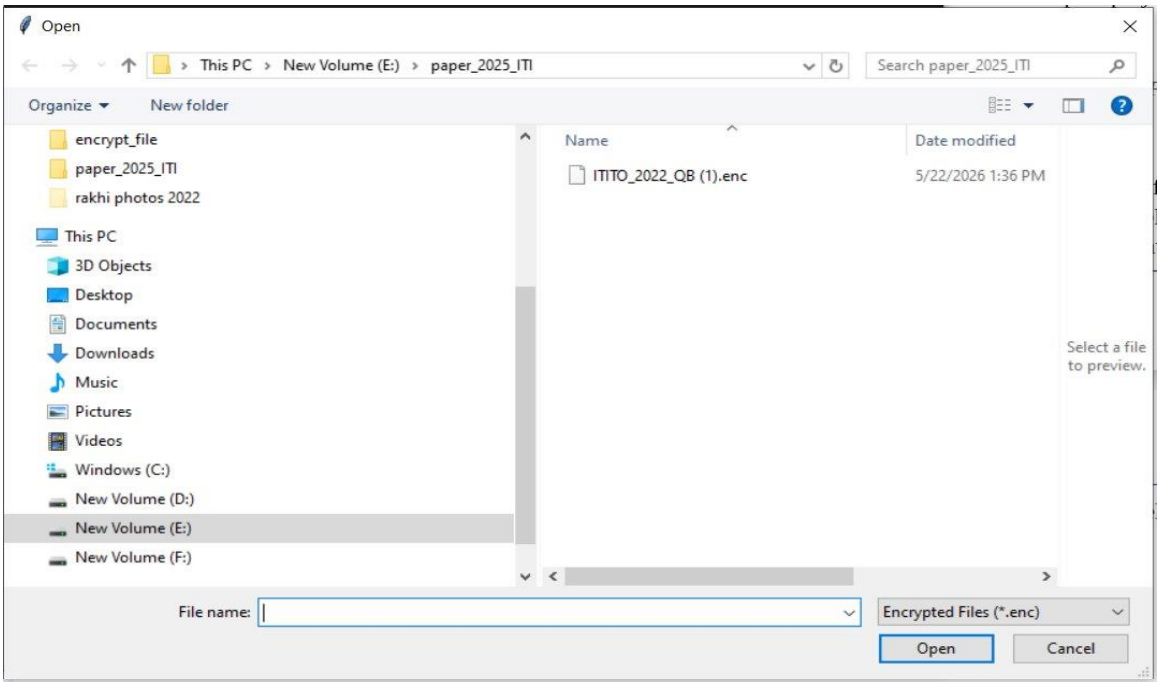

Figure 4: Encrypted File Selection Dialog — Step 1 of Decryption

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Windows Open dialog filtered to show only Encrypted Files (*.enc) — user selects the ITITO_2022_QB (1).enc file for decryption

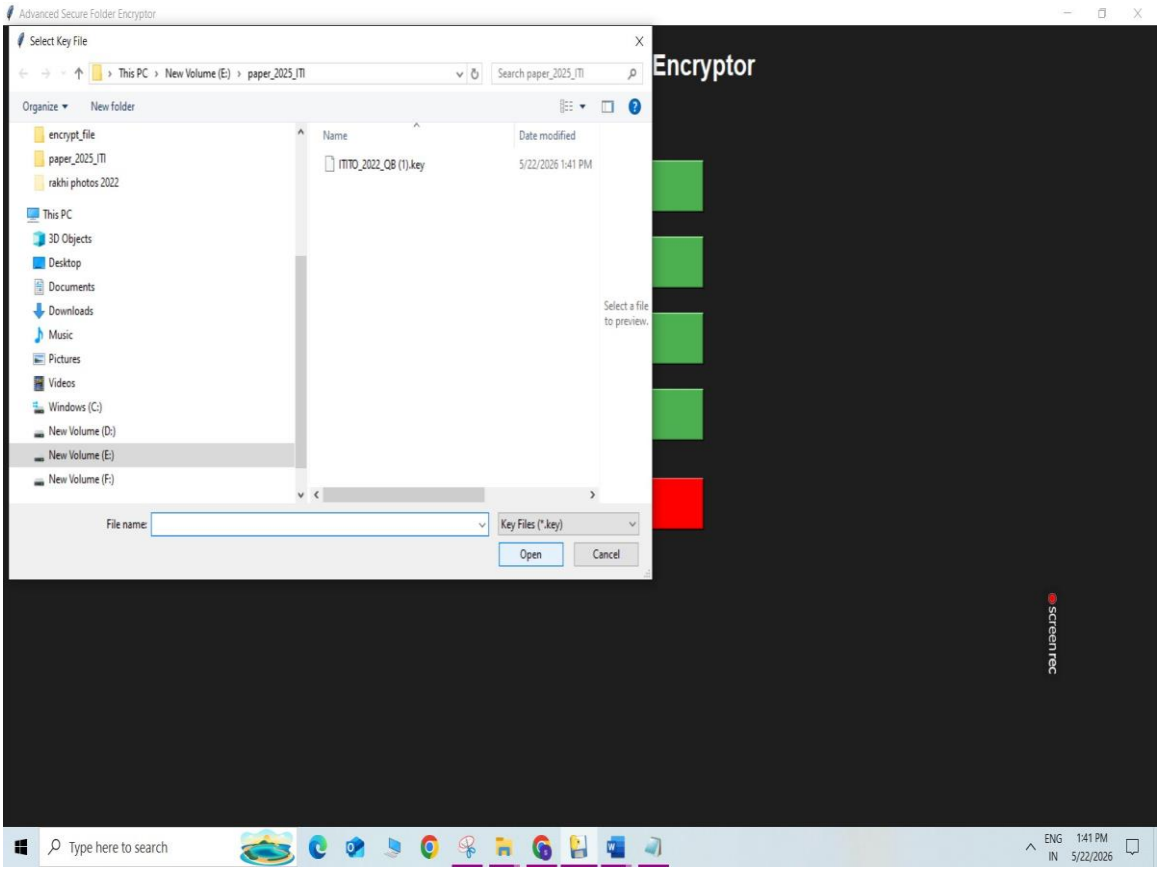

Figure 5: Key File Selection Dialog — Step 2 of Decryption

Select Key File dialog filtered to show only Key Files (*.key) — user selects the corresponding ITITO_2022_QB (1).key file

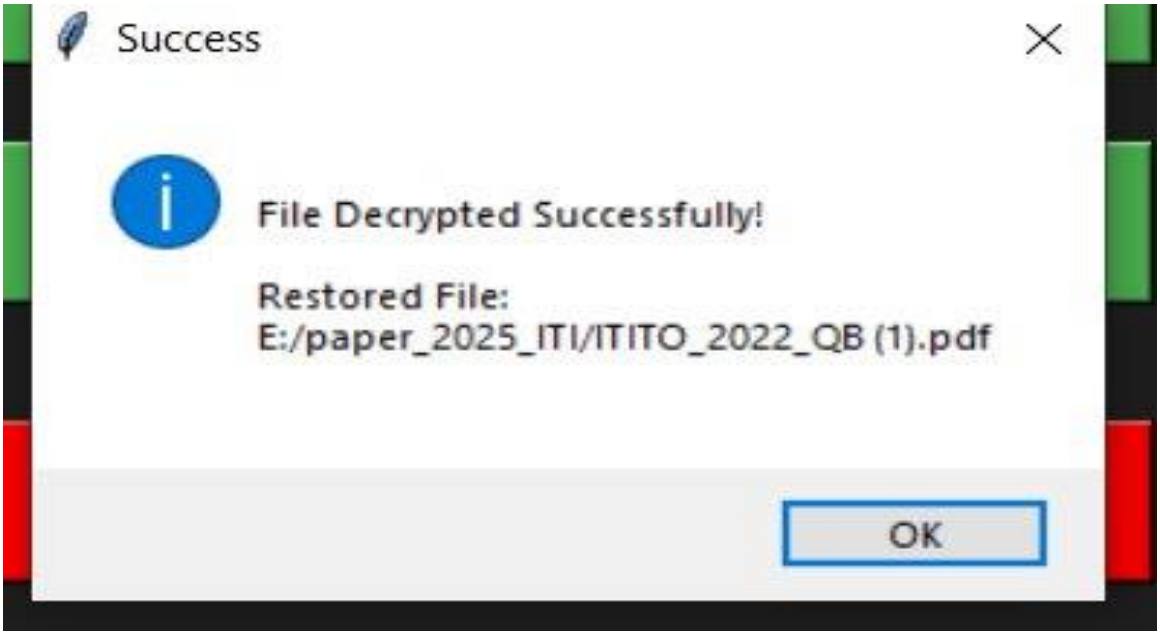

Figure 6: Decryption Success Dialog — Step 3 of Decryption

Success dialog confirming File Decrypted Successfully — the original file ITITO_2022_QB (1).pdf has been fully restored to its original path

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Folder Encryption Workflow

Clicking Encrypt Whole Folder opens a folder browser for selecting the target folder. After selection, the application performs three sequential steps: (1) compresses the entire folder into a temporary ZIP archive — the Compressing Folder progress dialog is displayed; (2) generates a unique Fernet key; (3) encrypts the ZIP archive. On completion, a success dialog shows the path to the single .enc file, the .key file path, and confirms the original folder has been removed.

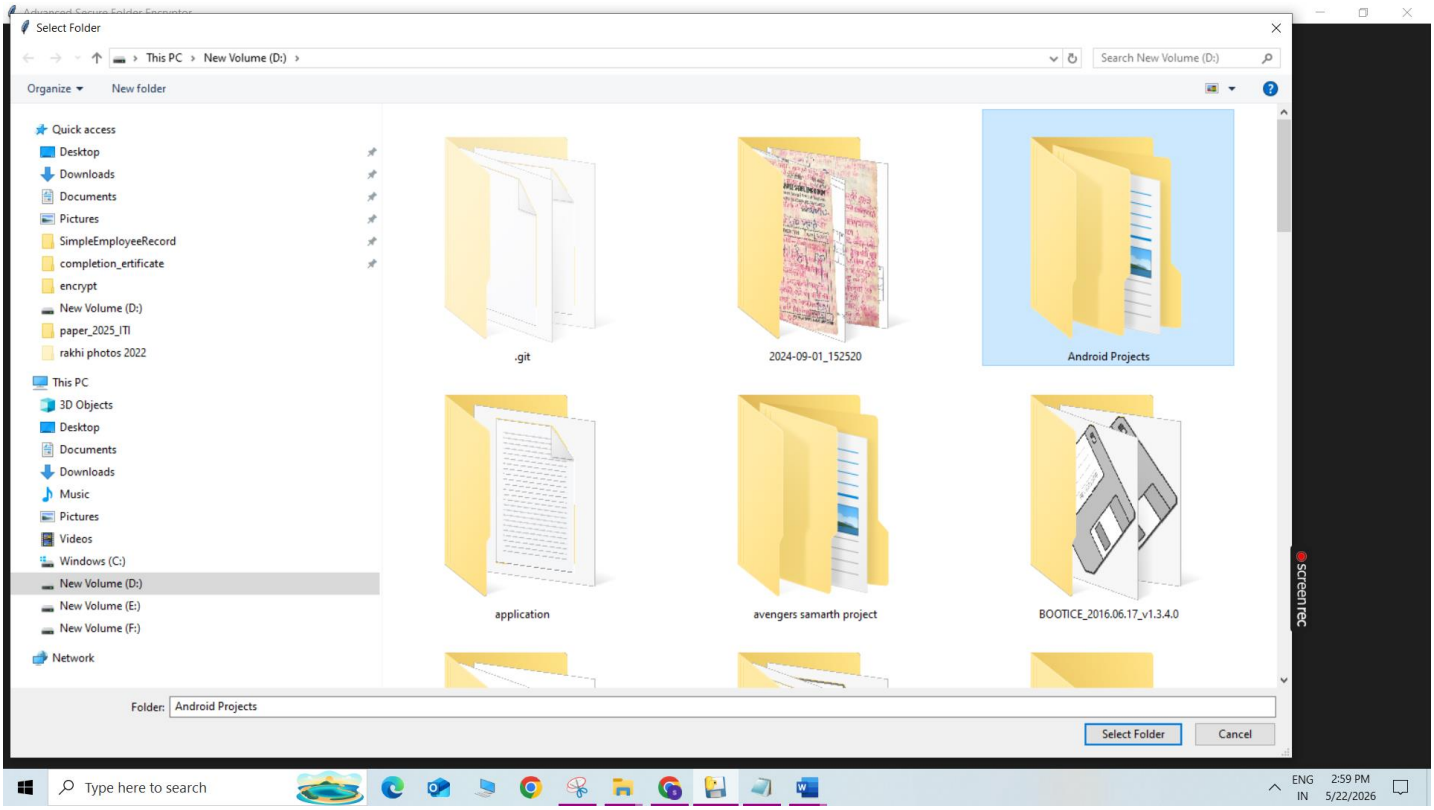

Figure 7: Folder Browser Dialog — Step 1 of Folder Encryption

Native Windows folder browser dialog for selecting the target folder to encrypt — folder 'Android Project' selected from New Volume (D:)

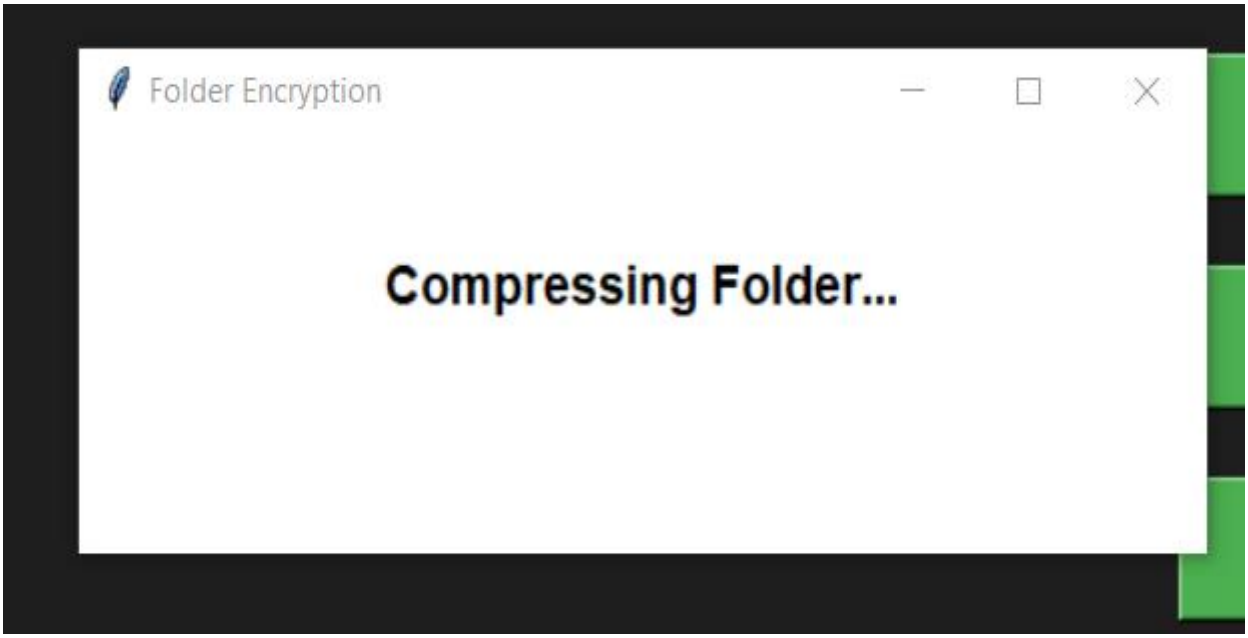

Figure 8: Folder Compression Progress — Step 2 of Folder Encryption

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Folder Encryption progress dialog displaying 'Compressing Folder...' during the ZIP compression stage before Fernet encryption is applied

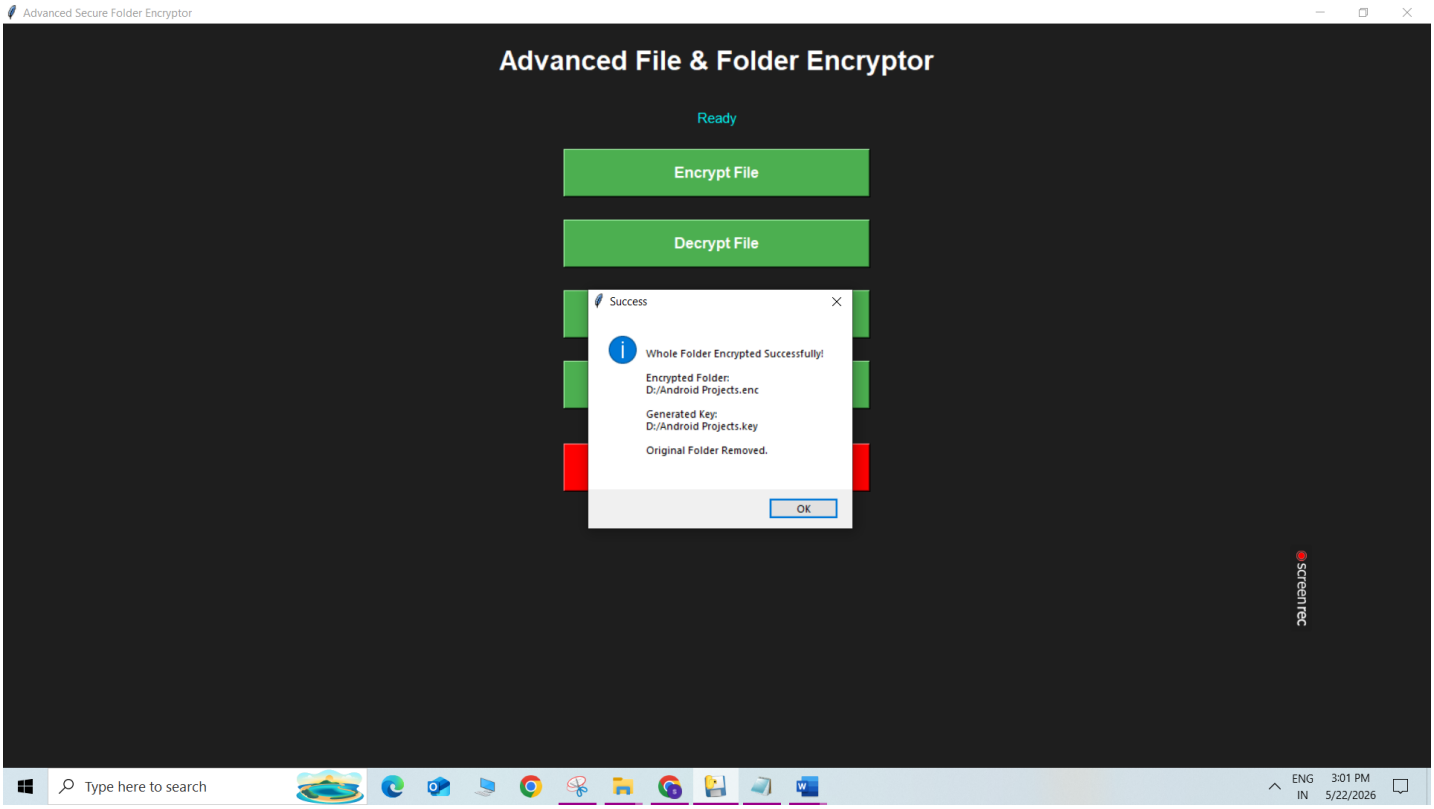

Figure 9: Folder Encryption Success Dialog — Step 3 of Folder Encryption

Success dialog confirming Whole Folder Encrypted Successfully — showing the encrypted folder path, generated key path, and confirmation that the Original Folder has been Removed

Folder Decryption Workflow

Clicking Decrypt Whole Folder opens two sequential dialogs — first to select the .enc folder file, then to select the .key file. The application decrypts the .enc file back to a ZIP archive, extracts all contents to restore the original folder structure, then removes the temporary files. A progress dialog shows 'Restoring Folder...' during extraction.

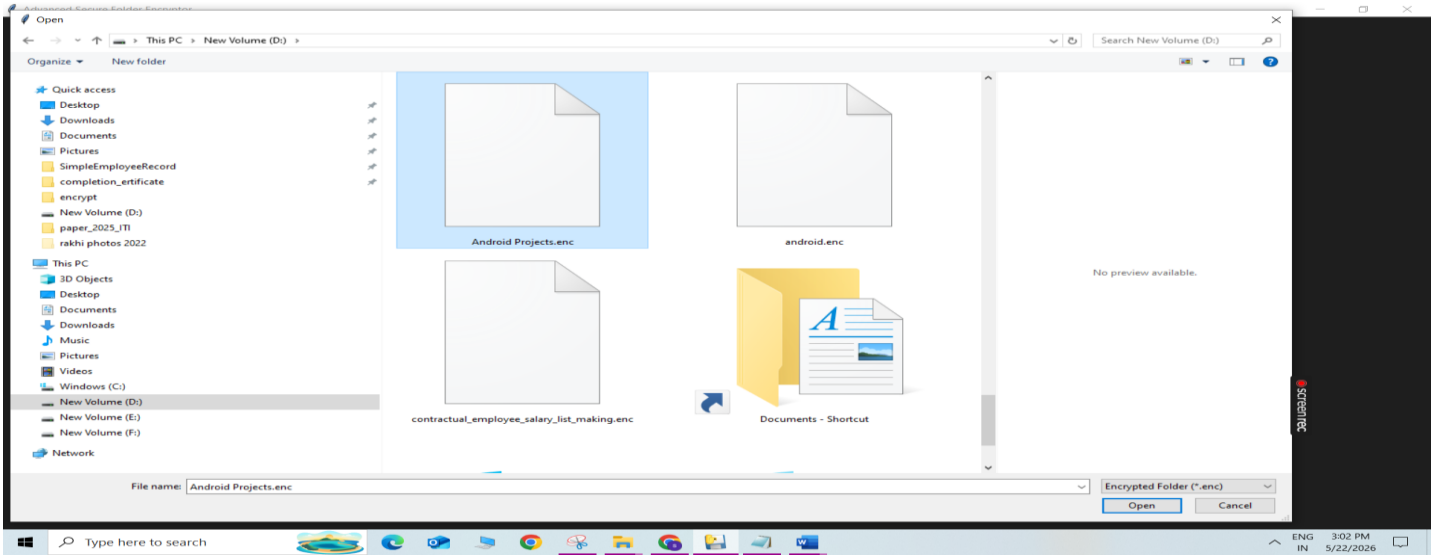

Figure 10: Encrypted Folder Selection Dialog — Step 1 of Folder Decryption

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Open dialog filtered to show only Encrypted Folder (*.enc) files — Android Projects.enc file selected for folder decryption

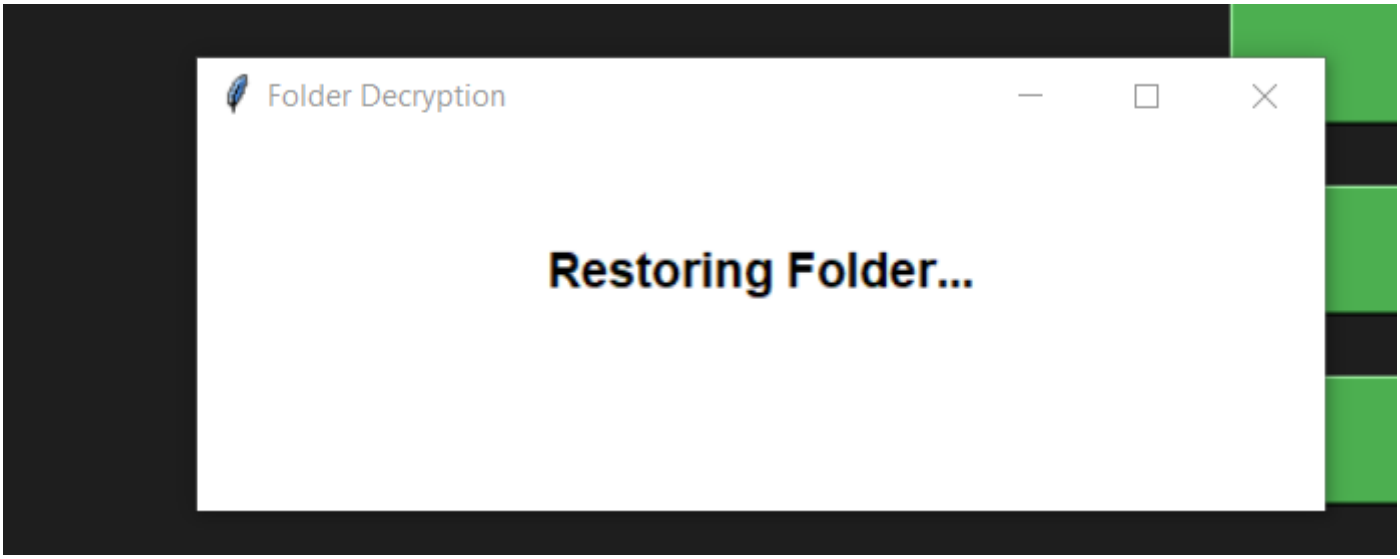

Figure 11: Folder Restoration Progress — Step 2 of Folder Decryption

Folder Decryption progress dialog displaying 'Restoring Folder...' while the ZIP archive is being extracted to reconstruct the original folder structure

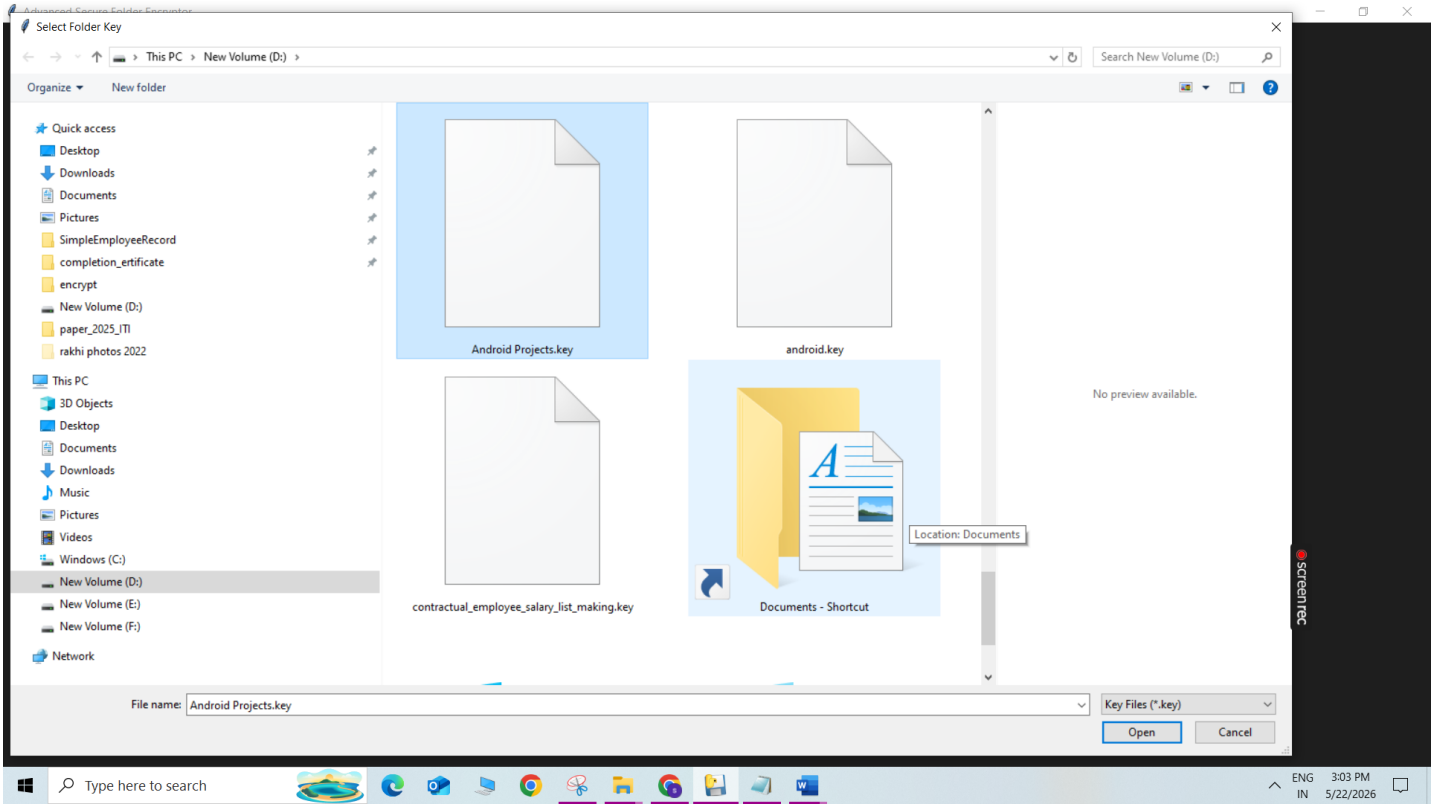

Figure 12: Folder Decryption Success Dialog — Step 3 of Folder Decryption

Success dialog confirming Folder Decrypted Successfully — showing the Restored Folder path (D:/Android Projects)

Exit Confirmation

Clicking the Exit button displays a yes/no confirmation dialog to prevent accidental closure of the application mid-operation. Clicking "Yes" closes the application; clicking "No" returns the user to the main window.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

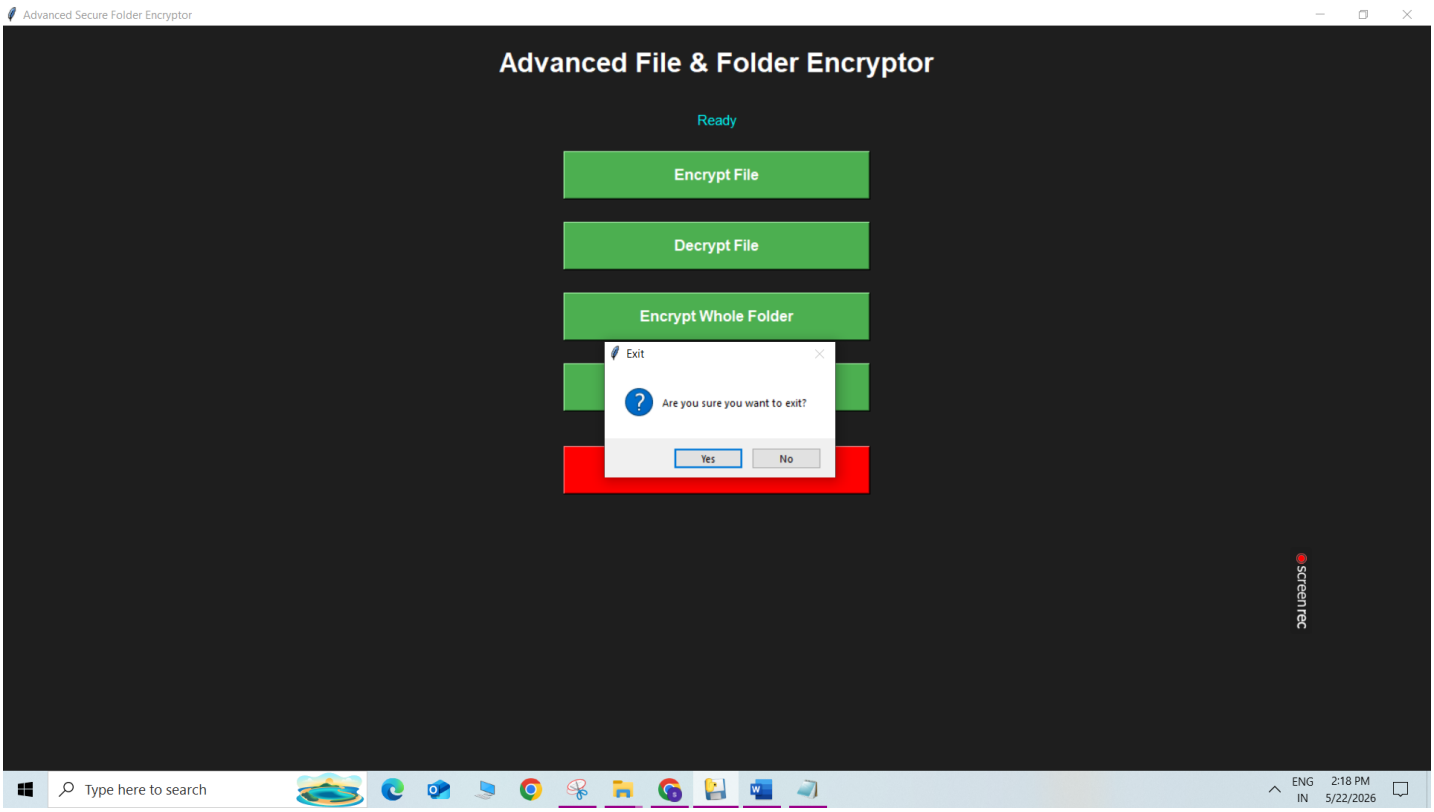

Figure 13: Exit Confirmation Dialog

Exit confirmation dialog with 'Are you sure you want to exit?' message — Yes closes the application, No returns to the main window

Application Features & Security Properties

The Secure File & Folder Encryptor provides the following capabilities and security guarantees:

Feature / Property	**Description**
File Encryption & Decryption	Encrypt/decrypt any file type (.docx, .jpg, .pdf, .exe, etc.) with one click
Folder Encryption & Decryption	Compress and encrypt entire folder trees into a single .enc file
Image Encryption	Images and media files encrypted using the same Fernet pipeline
Unique Key per Operation	Fernet.generate_key() issues a fresh 256-bit key every session
Integrity Verification	HMAC-SHA256 detects any ciphertext tampering before decryption
Automatic Plaintext Deletion	Original files removed immediately post-encryption
Extension Preservation	Original extension embedded in payload; restored on decryption
Progress Feedback	Real-time progress dialogs prevent appearance of application freezing
Offline Operation	No network dependency; keys never transmitted over a network
Windows Installer (.exe)	Distributable setup installer; no Python installation required

Table 3: Application Features and Security Properties

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

Benefits & Target Beneficiaries

The application delivers security, usability, and operational benefits. From a security perspective, end-to-end authenticated encryption ensures that even if an attacker obtains the .enc file, no information is recoverable without the key. Unique keys per file eliminate key-reuse attacks. Automatic deletion of plaintext prevents residual data exposure. From a usability standpoint, zero cryptographic knowledge is required, native Windows dialogs provide a familiar experience, and the Windows Setup .exe installer requires no Python environment.

Beneficiary Group	How They Benefit
Students & Researchers	Encrypt thesis drafts, research data, and academic files before sharing
Small & Medium Businesses	Protect employee records, financial reports, and trade secrets at rest
Healthcare Professionals	Encrypt patient records to comply with HIPAA data privacy requirements
Legal Professionals	Secure case files maintaining attorney-client privilege at the file level
Journalists & Activists	Protect source documents and investigative notes from surveillance
IT & Security Teams	Encrypt backup files, configuration exports, and sensitive logs
Educational Institutions	Encrypt exam papers and student data (FERPA/GDPR compliance)
Personal Home Users	Protect private photos, financial documents, and identity records

Table 4: Target Beneficiaries

Regulatory Compliance Alignment

The application aligns with major cybersecurity and data privacy frameworks, making it suitable for regulated industries:

- NIST SP 800-111 — Guide to Storage Encryption Technologies for End User Devices
- ISO/IEC 27001:2022 — Information Security Management: Asset Protection Controls
- GDPR Article 32 — Security of Processing: recommends encryption as a technical measure
- HIPAA Security Rule § 164.312(a)(2)(iv) — Encryption of Electronic Protected Health Information
- PCI-DSS Requirement 3.5 — Protect stored cardholder data using strong cryptography

10. CURRENT LIMITATIONS & FUTURE ROADMAP

10.1 Current Limitations

- Desktop-only implementation limited to Windows platform.
- Single-threaded execution may block GUI thread during large file operations.
- Key stored as plaintext .key file; no password-based key derivation (PBKDF2/Argon2).
- No audit logging, multi-user authentication, or cloud synchronization.
- No multi-file batch encryption in a single operation.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

- Key loss results in permanent data inaccessibility with no recovery mechanism.

Future Roadmap: Migration to Django / Flask

The planned evolution migrates from Tkinter to a full web application. The web version will use Django ORM for user account management, Celery for background task processing, Django REST Framework for an API layer, and Amazon S3 for encrypted file hosting. Future AI integrations include malware detection, suspicious file analysis, smart threat detection, user behavior monitoring, and AI-powered ransomware prevention.

Feature	Desktop (Current)	Web (Future)
Platform	Windows only	Any OS, any browser
Multi-user support	Single user	Multiple users with accounts
Key management	Manual .key file	Server-side key vault / password auth
Audit trail	None	Full database log
Team sharing	Not supported	Shared encrypted vaults
Installer required	Yes (Setup .exe)	No (browser-based)

Table 5: Desktop vs. Web Version Comparison

CONCLUSION

The Secure File & Folder Encryptor is an efficient, accessible, and cryptographically sound desktop application that addresses one of the most persistent challenges in personal and professional computing: protecting sensitive data at rest. By leveraging Python's cryptography library and the Fernet authenticated encryption scheme (AES-128-CBC + HMAC-SHA256), the application delivers strong data protection in a package requiring no technical expertise from the end user.

The application's design philosophy—one button, one operation, one unique key—mirrors the principle of least privilege and simplicity that cybersecurity best practices advocate. By automatically removing plaintext originals, embedding extension metadata within the ciphertext, and enabling full folder-tree encryption via ZIP compression, the tool addresses a broad range of real-world data protection scenarios.

Beneficiaries span virtually every category of computer user: from students protecting thesis drafts and freelancers securing client deliverables, to healthcare providers safeguarding patient records and journalists protecting source documents. The alignment with GDPR, HIPAA, PCI-DSS, and NIST guidelines makes the application especially relevant in regulated industries where encryption is a legal obligation. The planned migration to Django/Flask will transform this capable desktop tool into a scalable, multi-user, cross-platform encryption service.

REFERENCES

1. Python Cryptography Authority. (2024). cryptography — Fernet (Symmetric Encryption). PyPI / Read the Docs. https://cryptography.io/en/latest/fernet/
2. National Institute of Standards and Technology. (2007). NIST SP 800-111: Guide to Storage Encryption Technologies for End User Devices. U.S. Department of Commerce.
3. International Organization for Standardization. (2022). ISO/IEC 27001:2022 — Information Security Management Systems.
4. European Parliament. (2016). General Data Protection Regulation (GDPR) — Article 32: Security of Processing. Official Journal of the European Union.

ISSN No. 2321-2705 | DOI: 10.51244/IJRSI |Volume XIII Issue V May 2026

5. U.S. Department of Health and Human Services. (2013). HIPAA Security Rule — § 164.312(a)(2)(iv). Federal Register.
6. Python Software Foundation. (2024). tkinter — Python interface to Tcl/Tk. https://docs.python.org/3/library/tkinter.html
7. Python Software Foundation. (2024). zipfile — Work with ZIP archives. https://docs.python.org/3/library/zipfile.html
8. Payment Card Industry Security Standards Council. (2022). PCI DSS v4.0 — Requirement 3: Protect Stored Account Data.
9. Django Software Foundation. (2024). Django Documentation. https://docs.djangoproject.com/
10. Pallets Projects. (2024). Flask Documentation. https://flask.palletsprojects.com/
11. 13. GITHUB REPOSITORY AND SOFTWARE DOWNLOAD LINKS
12. The complete source code, setup installer, and project resources of the Secure File & Folder Encryptor application are publicly available on GitHub. Researchers, developers, and end users are encouraged to explore the repository, review the implementation, and contribute to future enhancements.
13. 13.1 GitHub Repository
14. The full project source code is hosted at the following GitHub repository:
15. Repository: https://github.com/shrutidalela/secure-file-folder-encryptor
16. 13.2 Software Download (Windows Setup Installer)
17. The Windows setup installer (.exe) for version 1.0 of the application can be downloaded from the GitHub Releases page:
18. Release Page: https://github.com/shrutidalela/secure-file-folder-encryptor/releases/tag/v1.0
19. 13.3 Direct Setup Download
20. A direct download link for the SecureEncryptorSetup.exe Windows installer is provided below. No Python installation or additional dependencies are required:
21. Direct Download: https://github.com/shrutidalela/secure-file-folder-encryptor/releases/download/v1.0/SecureEncryptorSetup.exe
22. 13.4 Installation Instructions
23. Download SecureEncryptorSetup.exe from the direct link above.
24. Run the installer on any Windows 10/11 system.
25. Follow the on-screen setup wizard to complete installation.
26. Launch Secure File & Folder Encryptor from the Start Menu or Desktop shortcut.
27. Use the Encrypt File, Decrypt File, Encrypt Whole Folder, or Decrypt Whole Folder buttons.
28. Store your generated .key file in a secure, separate location from the .enc file.