Secure File Encryption and Decryption Tool

A command-line tool for securely encrypting and decrypting files using AES-256, ensuring confidentiality and integrity.

Installation

1. **Prerequisites**:
   - Python 3.8 or higher.
   - Install dependencies:

```
pip install cryptography
```

2. **Setup**:
   - Place the script secure_file_tool.py in your working directory.

Usage

Run the tool via the command line:

```
python secure_file_tool.py <mode> <file> <password> <output>
```

Modes:

- encrypt: Encrypt a file.
- decrypt: Decrypt a file.

Arguments:

- <file>: Path to the input file.
- <password>: Password for encryption or decryption.
- <output>: Path to save the encrypted or decrypted file.

Examples

Encrypt a File

```
python secure_file_tool.py encrypt secret.txt mypassword encrypted.bin
```

- Encrypts secret.txt with the password mypassword and saves it as encrypted.bin.

Decrypt a File

```
python secure_file_tool.py decrypt encrypted.bin mypassword decrypted.txt
```

- Decrypts encrypted.bin with the password mypassword and saves it as decrypted.txt.

Features

1. **Encryption and Decryption**:
   - AES-256 encryption ensures data confidentiality.
2. **Password-Based Key Derivation**:
   - Uses PBKDF2 with a unique salt for each file.
3. **Integrity Check**:
   - HMAC ensures file integrity and detects tampering.
4. **Compression**:
   - Compresses files before encryption to save space.
5. **Brute-Force Protection**:
   - Limits to 5 incorrect decryption attempts with a 5-minute lockout.
6. **Cross-Platform**:
   - Compatible with Windows, macOS, and Linux.

Error Handling

- **Incorrect Password**:
  - Outputs: HMAC verification failed. File integrity compromised.
- **Too Many Attempts**:
  - Outputs: Too many failed attempts. Please try again later.

Report

1. Encryption Method

The tool uses **AES-256** in Cipher Feedback (CFB) mode, which ensures:

- Strong encryption with a 256-bit key.
- Compatibility with files of any size.

2. Key Derivation

PBKDF2 (Password-Based Key Derivation Function 2) is implemented:

- Uses **SHA-256** as the hash function.
- Derives a secure 256-bit key from the user's password.
- Includes a unique 16-byte **salt** to prevent precomputed attacks.
- Iterations set to 100,000 to resist brute-force attempts.

3. Data Integrity

The tool integrates **HMAC (Hash-based Message Authentication Code)**:

- Validates the authenticity of the encrypted data.
- Uses the same derived key as the encryption process.

4. Defense Mechanisms

- **Brute-Force Protection**:
    - Limits incorrect decryption attempts to 5.
    - Enforces a 5-minute lockout after exceeding the limit.
- **Secure Key Management**:
    - Keys are stored in memory temporarily during encryption/decryption.
    - No passwords or keys are saved to disk.

5. Cross-Platform Compatibility

- The tool uses standard libraries to ensure compatibility across all major operating systems.

6. Future Improvements

1. **Graphical User Interface (GUI)**:
    - Provide a user-friendly interface for non-technical users.
2. **Hybrid Encryption**:
    - Use RSA to encrypt AES keys for secure key sharing.
3. **File Shredding**:
    - Securely delete plaintext files after encryption.