Password Manager: Information Security Project Report

Team Members:

Zohra Amna (2022-SE-03)

Muhammad Noman (2022-SE-31)

Syed Abdul Ali Shah (2022-SE-36)

Rana Muhammad Saifullah (2022-SE-39)

Supervisor:

Mam Anam Iftikhar

Institution/Department:

[Institution Name], Department of Software Engineering

Date of Submission:

May 4, 2025

Abstract

The Password Manager is a desktop application designed to securely store and manage website credentials. Developed using Python and Tkinter, it employs AES-256-GCM encryption and a manual SHA-256 implementation to hash the master password, ensuring robust security. Key features include user authentication, password generation, storage, and retrieval. The project has been converted to an executable for easy distribution and use. It successfully delivers a user-friendly interface and secure data handling, though it faces limitations with weak master passwords and file-based storage.

Contents

1 Introduction

Passwords are a critical component of digital security, yet managing them securely remains a challenge due to weak passwords and unsafe storage practices. The Password Manager addresses this by providing a secure, user-friendly desktop application to store and manage website credentials. This project is important because it mitigates risks associated with password reuse and exposure, common causes of data breaches. Real-world applications include personal credential management and small-scale enterprise use. The scope covers user authentication, password generation, AES-256-GCM encryption, credential retrieval, and conversion to an executable for easy deployment, but excludes multi-user support and cloud integration.

2 Objectives

The project aimed to achieve the following:

- Develop a secure desktop application for managing website credentials.
- Implement user authentication with SHA-256 hashed master passwords.
- Provide strong password generation (12–18 characters with mixed characters).
- Encrypt stored passwords using AES-256-GCM with the master password hash as the key.
- Enable secure password retrieval and clipboard functionality.
- Ensure a user-friendly interface using Tkinter.
- Convert the application to an executable for easy distribution and use.

3 Literature Review / Related Work

Existing password managers like LastPass and 1Password offer robust features, including cloud storage and cross-platform support. However, they rely on proprietary systems, raising privacy concerns for some users. Open-source alternatives like KeePass use local storage but often lack intuitive interfaces. These systems typically employ AES encryption and key derivation functions (e.g., PBKDF2), but weak master passwords remain a vulnerability. Our project addresses the gap of providing a simple, open-source desktop solution with strong encryption and an executable format for accessibility, though it lacks key stretching and secure file storage compared to commercial options.

4 System Analysis and Design

4.1 System Architecture

The Password Manager is a standalone desktop application with a client-side architecture. It consists of:

- **UI Layer**: Tkinter-based graphical interface for user interaction.
- **Logic Layer**: Python functions for authentication, encryption, and password management.
- **Storage Layer**: JSON files (`users.json` for hashed master passwords, `<email>_passwords.json` for encrypted credentials).

4.2 Data Flow Diagram (DFD)

1. **User Input**: Email and master password entered via UI.
2. **Authentication**: Master password hashed (SHA-256) and verified against `users.json`.
3. **Password Management**: Credentials encrypted (AES-256-GCM) and stored in `<email>_passwords.json`.
4. **Retrieval**: Website name searched, credentials decrypted, and displayed.

4.3 Threat Models

- **Weak Master Password**: Susceptible to brute-force attacks due to no key stretching.
- **File Exposure**: Unencrypted JSON files vulnerable if the system is compromised.
- **Memory Attacks**: Master password hash stored in memory during the session.

5 Technology Stack

- **Programming Language**: Python 3.12
- **Libraries**:
  - Tkinter (GUI framework)
  - Cryptography (AES-256-GCM encryption)
  - PyInstaller (for converting to executable)
- **Tools**: Manual SHA-256 implementation for hashing
- **Files**: `logo.png` for UI branding

6 Implementation Details

The application was developed in Python, with Tkinter for the GUI and the `cryptography` library for encryption. It was converted to an executable using PyInstaller for easy distribution and use without requiring a Python environment. Key modules include:

1. **User Authentication**:
    - `create_account()`: Hashes master password using `sha256_manual()` and stores it in `users.json`.
    - `login()`: Verifies hashed master password and grants access.
2. **Password Generation**:
    - `gen_password()`: Generates random 12–18 character passwords with letters, numbers, and symbols, copied to clipboard.
3. **Password Storage**:
    - `encrypt_password()`: Uses AES-256-GCM with the master password hash as the key, storing ciphertext, nonce, and tag in `<email>_passwords.json`.
    - `save_password()`: Saves encrypted credentials.
4. **Password Retrieval**:
    - `search_password()`: Decrypts and displays credentials using `decrypt_password()`.
5. **UI Setup**:
    - `setup_main_ui()`: Creates the main interface with fields for website, email, password, and buttons for actions.
6. **Executable Conversion**:
    - Used PyInstaller to package the Python script, dependencies, and `logo.png` into a standalone executable, enabling deployment on systems without Python.

Screenshots: Login window with email and password fields; main interface with password generation and search options.

7 Testing and Validation

- **Unit Testing**: Tested individual functions (`sha256_manual`, `encrypt_password`, `decrypt_password`) for correctness.
- **Vulnerability Scans**: Manual checks for SQL injection (not applicable) and file access vulnerabilities.
- **Penetration Testing**: Simulated brute-force attacks on weak master passwords, confirming vulnerability.
- **Executable Testing**: Verified the executable runs correctly on Windows and Linux, with all features intact.
- **Performance Results**:

  - Password generation: <0.1 seconds.
  - Encryption/decryption: <0.5 seconds for typical credentials.
  - Executable startup: <2 seconds.
  - Compliance: Aligns with OWASP password storage guidelines but lacks key derivation.

8 Security Features

- **Encryption**: AES-256-GCM for website passwords, ensuring confidentiality and integrity.
- **Hashing**: Manual SHA-256 for master password, stored in `users.json`.
- **Authentication**: Master password verification before access.
- **Vulnerability Handling**: No plaintext passwords stored; unique nonces prevent replay attacks.
- **Compliance**: Partial OWASP compliance (secure storage but no key stretching).

9 Challenges and Limitations

- **Challenges**:
  - Implementing manual SHA-256 was complex and error-prone.
  - Ensuring AES-GCM nonce uniqueness required careful design.
  - Packaging with PyInstaller required resolving dependency conflicts for the executable.
- **Limitations**:
  - No key stretching (e.g., PBKDF2), making weak master passwords vulnerable.
  - JSON files stored unencrypted on disk.
  - No master password strength validation.
  - Single-user focus, no session timeouts.

10 Future Work

- Implement PBKDF2 or Argon2 for key derivation to strengthen weak passwords.
- Use a secure database or encrypted files for storage.

- Add password strength validation during account creation.
- Introduce session timeouts and multi-user support.
- Enhance UI with cross-platform compatibility (e.g., PyQt).
- Optimize executable size and startup time.

11 Conclusion

The Password Manager successfully delivers a secure, user-friendly solution for managing website credentials using AES-256-GCM encryption and SHA-256 hashing. Converted to an executable using PyInstaller, it ensures easy deployment without requiring a Python environment. It provides essential features like password generation, storage, and retrieval, with an intuitive Tkinter interface. Despite limitations like weak password vulnerability and file-based storage, the project demonstrates practical cryptographic techniques and serves as a foundation for future enhancements. Its real-world value lies in enabling secure credential management for individuals and small teams.

12 References

1. Python Software Foundation, "Python 3.12 Documentation," 2023. [Online]. Available: `https://docs.python.org/3.12/`
2. "Cryptography Library Documentation," 2023. [Online]. Available: `https://cryptography.io/`
3. OWASP, "Password Storage Cheat Sheet," 2023. [Online]. Available: `https://cheatsheetseries.owasp.org/`
4. National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," NIST FIPS 197, 2001.
5. D. J. Bernstein, "SHA-256 Specification," 2004. [Online]. Available: `https://csrc.nist.gov/`
6. PyInstaller Development Team, "PyInstaller Documentation," 2023. [Online]. Available: `https://pyinstaller.org/`

Instructions for Converting to Word

To obtain a Word document from this LaTeX source:

1. Copy the LaTeX code into Overleaf (www.overleaf.com).
2. Compile the document using PDFLaTeX to generate a PDF.
3. Download the PDF.

4. Use a PDF-to-Word converter (e.g., Adobe Acrobat, pdf2docx.com, or Smallpdf) to convert the PDF to a .docx file.

5. Review the .docx file for formatting consistency and make minor adjustments if needed.