CADT
IDT វិទ្យាស្ថានបច្ចេកវិទ្យាឌីជីថល
Institute of Digital Technology

Department of Telecom and Networking G1 | Year 3
Course: Cryptography
Lecturer: Meas Sothearath
Student's name: Chhy Bunhouy

Topic: Password Manager With Cryptography

I. Introduction / Background

Overview / Goal

The goal of this project is to build a secure password manager that helps users safely store and manage their passwords. Instead of remembering dozens of passwords for different websites and applications, a user only needs to remember a single master password. All other credentials are securely encrypted and stored in a local vault, protecting them from unauthorized access.

This project is implemented as a command-line interface (CLI) application for simplicity and clarity. Despite its simplicity, the application demonstrates the complete workflow of a real-world password manager, including secure authentication, encryption, storage, and retrieval of sensitive data.

Problems and Solutions

Problem 1: Password reuse and weak passwords
Many users reuse the same passwords or choose weak passwords, which significantly increases the risk of account compromise.

Solution: The application provides a strong password generator that creates random, complex passwords using cryptographically secure randomness.

Problem 2: Insecure plaintext storage
Storing passwords in plaintext files or notes allows attackers to steal all credentials if access is gained.

Solution: All passwords are encrypted using AES-GCM, and encryption keys are derived from the master password using PBKDF2 with a unique salt.

Problem 3: Difficulty remembering many passwords
Managing multiple passwords for different services is inconvenient for users.

Solution: The password manager acts as a centralized vault, requiring users to remember only one master password.

Problem 4: Vault file theft
Encrypted vault files can be copied or stolen by attackers.

Solution: Even if the vault file is stolen, it remains unreadable without the correct master password and derived encryption key.

Motivation

This project was developed to address a common and practical cybersecurity problem that affects almost every internet user. Password-based attacks remain one of the most common attack vectors. By building a password manager, this project combines software development skills with applied cryptography concepts learned in class.

The CLI-based design allows easier testing and understanding of the internal security mechanisms, making the project both educational and extensible.

Cryptographic Concepts Used

- **AES-GCM (Advanced Encryption Standard – Galois/Counter Mode):** Provides confidentiality and integrity for stored passwords.
- **PBKDF2 (Password-Based Key Derivation Function 2):** Derives a strong encryption key from the master password using salting and multiple iterations.
- **Salt and Nonce:** Salt protects against precomputed attacks, while nonces ensure unique encryption for each entry.
- **Secure Random Generation:** Uses Python's `secrets` module to generate unpredictable passwords and cryptographic values.
- **Base64 Encoding:** Converts binary encrypted data into a text format suitable for file storage.

II. System Design / Architecture

The system follows a modular architecture that separates cryptographic operations, storage management, and user interaction. This separation improves maintainability and security.

High-Level Architecture

1. **User Interface (CLI):** Handles user input and menu navigation.
2. **Key Management Module:** Manages master password handling, key derivation, and lockout protection.
3. **Cryptographic Engine:** Performs AES-GCM encryption and decryption.
4. **Vault Manager:** Controls password entry creation, retrieval, and deletion.

5. **Secure Storage:** Stores encrypted password blobs in a local file.

Data Flow

1. User enters the master password.
2. PBKDF2 derives a 256-bit encryption key using a stored salt.
3. Password entries are converted to JSON format.
4. Data is encrypted using AES-GCM with a random nonce.
5. Encrypted data, nonce, and authentication tag are stored in the vault file.
6. During retrieval, the same key and nonce are used to decrypt the data.

Each password entry is encrypted independently, reducing the impact of data corruption and allowing selective deletion.

III. Implementation Details

The application is implemented in Python and uses the `cryptography` library to ensure secure and reliable cryptographic operations.

Key Components

KeyManager (`key_manager.py`):

- Generates a random salt for each vault.
- Derives a 32-byte encryption key using PBKDF2-HMAC-SHA256 with 200,000 iterations.
- Implements password attempt limits and temporary lockout to prevent brute-force attacks.

CryptoEngine (`crypto_engine.py`):

- Wraps AES-GCM encryption and decryption.
- Ensures both confidentiality and integrity of stored data.

VaultManager (`vault_manager.py`):

- Encrypts individual password entries.
- Decrypts entries only when required and only in memory.
- Supports add, view, list, delete, and generate password operations.

Vault Storage (`vault_storage.py`):

- Stores each encrypted entry as a JSON object per line.
- Separates nonce, ciphertext, and authentication tag.

- Uses Base64 encoding for safe file storage.

Password Generator (`password_generator.py`):

- Generates strong passwords using cryptographically secure randomness.
- Evaluates password strength for user feedback.

User Interface (`menu.py`):

- Provides a clear CLI menu for interacting with the vault.
- Uses secure input methods to hide sensitive data.

IV. Usage Guide

This section explains in detail how to install, configure, and use the Secure Password Manager application.

System Requirements

- Operating System: Windows, Linux, or macOS
- Python Version: Python 3.9 or higher
- Internet Connection: Not required (offline application)

Dependencies Installation

Before running the application, required dependencies must be installed.

Ensure Python is installed:
python --version

1.

Install required libraries using:
pip install -r requirements.txt

2.

The main dependency is the `cryptography` library, which provides secure cryptographic primitives.

Project Setup

1. Extract the project folder.

Navigate to the `src` directory:
cd password_manager/src

2.
3. Ensure the following directory structure exists:
    - `data/` (created automatically on first run)
    - `master.json` (stores master key metadata)
    - `vault.data` (stores encrypted password entries)

Running the Application

Start the application using:

python main.py

On launch, the program displays a secure password manager header and checks whether a vault already exists.

First-Time Setup (Create New Vault)

If no vault exists:

1. The user is prompted to create a master password.
2. Password strength is evaluated and displayed.
3. The user confirms the master password.
4. A random salt is generated.
5. A 256-bit encryption key is derived using PBKDF2.
6. The master key hash and salt are stored securely in `master.json`.

Important: The master password cannot be recovered if forgotten.

Unlocking an Existing Vault

If a vault already exists:

1. The user is prompted to enter the master password.
2. The system derives a key using the stored salt.

3. The derived key is compared with the stored master hash.
4. If authentication succeeds, the vault is unlocked.
5. After multiple failed attempts, the vault is temporarily locked.

This mechanism prevents brute-force password attacks.

Main Menu Options

Once unlocked, the following menu options are available:

1. **Add New Password**
    - Enter website/application name and username.
    - Choose to enter a password manually or generate a strong password.
    - The password is encrypted and stored in the vault.
2. **List Saved Passwords**
    - Displays a list of stored website names and usernames.
    - Passwords are not revealed in this view.
3. **Show Password**
    - Select a website to reveal stored credentials.
    - Data is decrypted temporarily in memory and displayed.
4. **Delete Password**
    - Select an entry to remove from the vault.
    - Confirmation is required before deletion.
5. **Generate Random Password**
    - Generates a strong password of user-defined length.
    - Displays password strength and length.
6. **Exit**
    - Locks the vault and safely exits the application.

Result and Output

- All passwords are stored only in encrypted form.
- Plaintext passwords exist only in memory during usage.
- The vault file remains secure even if copied or stolen.
- The system provides a balance between security and usability.

V. Conclusion and Future Work

This project demonstrates the effective use of cryptographic techniques to secure sensitive user data. By combining strong encryption, secure key derivation, and careful storage design, the password manager protects credentials even if the vault file is compromised.

Future Work

- Implement a web-based version of the password manager.
- Store encrypted data in a SQL database instead of local files.
- Add user authentication using email (e.g., Gmail login).
- Integrate multi-factor authentication (MFA) for additional security.
- Deploy the system securely on a cloud platform.

VI. References

1. Python Software Foundation, *Python Cryptography Library Documentation*.
2. OWASP Foundation, *OWASP Password Storage Cheat Sheet*, 2023.
3. RFC 4106 – AES-GCM Specification
4. RFC 8018 – PBKDF2 Specification