Encrypted Password Manager – Project Report

I. Introduction / Background

With the increasing use of online services, users are required to manage many usernames and passwords. Reusing passwords or storing them in plain text files is insecure and exposes users to risks such as data breaches, credential theft, and unauthorized access. Therefore, a secure way to store and manage passwords is essential.

The goal of this project is to design and implement a **local encrypted password manager** that securely stores user credentials in an encrypted vault file. The system ensures that sensitive data is protected even if the vault file is accessed by an attacker.

Problems Addressed

- Storing passwords in plain text is insecure
- Weak password protection against brute-force attacks
- Lack of encryption and integrity verification

Proposed Solution

This project uses modern cryptographic techniques to secure stored data: - **Argon2id** for password-based key derivation - **AES-256-GCM** for authenticated encryption - A local encrypted vault file (`vault.enc`)

Only users with the correct master password can decrypt and access stored credentials.

Related Cryptographic Concepts

- **Key Derivation Function (KDF):** Converts a human password into a strong cryptographic key
- **Salt:** Random data added to passwords to prevent rainbow-table attacks
- **Authenticated Encryption:** Ensures confidentiality and integrity of data

---

II. System Design / Architecture

The system is divided into three main components:

1. **User Interface (CLI)** – Handles user interaction
2. **Vault Manager** – Manages file storage and retrieval
3. **Cryptographic Module** – Handles encryption and decryption

Data Flow Overview

1. User enters a master password
2. The password is processed using Argon2id to derive a secure key
3. Password data is encrypted using AES-GCM
4. Encrypted data and salt are stored in `vault.enc`

5. During access, the same password regenerates the key to decrypt data

Encryption Flow

- Generate random salt
- Derive encryption key from master password
- Generate random nonce
- Encrypt data using AES-GCM
- Store salt + encrypted data

III. Implementation Details

Programming Language

- **Python 3**

Key Libraries Used

- `cryptography` – AES-GCM encryption
- `argon2-cffi` – Argon2 key derivation
- `json` – Data serialization
- `getpass` – Secure password input
- `os` – Secure random number generation

Key Components

1. Key Derivation (`derive_key`)

- Uses Argon2id
- Protects against brute-force and GPU attacks
- Produces a 256-bit encryption key

2. Encryption (`encrypt_data`)

- Uses AES-256-GCM
- Generates a random nonce
- Provides confidentiality and integrity

3. Decryption (`decrypt_data`)

- Verifies data integrity
- Fails if password or data is incorrect

4. Vault Storage

- Passwords stored as a Python dictionary
- Converted to JSON before encryption
- Saved as a binary encrypted file

Cryptographic Methods Used

- **Argon2id:** Password-based key derivation
- **AES-256-GCM:** Symmetric authenticated encryption

IV. Usage Guide

Dependencies

Install required libraries:

```
pip install cryptography argon2-cffi
```

How to Run the Program

1. Ensure all files (`main.py`, `vault.py`, `crypto_utils.py`) are in the same directory
2. Run the program:

   ```
   python main.py
   ```

3. Enter a master password
4. Choose options from the menu

Program Options

- Add a new password
- View stored passwords
- Delete an existing password
- Exit the program

Example Result

- Vault file created: `vault.enc`
- File contents are unreadable without the master password
- Correct password successfully decrypts stored credentials

V. Conclusion and Future Work

Conclusion

This project successfully demonstrates the implementation of a secure local password manager using modern cryptographic standards. By combining Argon2id and AES-256-GCM, the system ensures strong protection against common attacks such as brute-force attempts and data tampering.

Future Improvements

- Add password strength validation
- Implement clipboard auto-clear for copied passwords
- Add password search and categorization
- Implement GUI interface
- Support backup and recovery options

VI. References

1. RFC 9106 – Argon2 Password Hashing
2. NIST SP 800-38D – Galois/Counter Mode (GCM)
3. Cryptography.io Documentation
4. OWASP Password Storage Cheat Sheet
5. Python Official Documentation