CADT
IDT វិទ្យាស្ថានបច្ចេកវិទ្យាឌីជីថល
Institute of Digital Technology

DEPARTMENT OF

TELECOMS AND NETWORKING

Specialize in Cybersecurity

Course Title: **CRYPTOGRAPHY**

Term 1 | Year 3

Project Title:

Secure Password Manager using bcrypt and AES Encryption

Lect. Mr. Meas Sothearath

Submitted by: Tun Monireaksa , IDTB100327 ,Group 2

Submission date:20th December 2025

TABLE OF CONTENT

I. Introduction / Background

1.1 Overview of The Project Goal

The goal of this project is to design and implement a secure password manager that allows users to safely store and retrieve their online account credentials. The system uses a master password for authentication and applied modern cryptographic techniques to ensure that all sored passwords remain encrypted , protected , and inaccessible to unauthorized users.

The project demonstrates how secure systems project confidential information such as: Website logins , Application credentials , Personal passwords. It is look like or mimics the core security design of real password managers like LastPass, 1Password…, but in a simplified, educational form.

1.2 Problem Statement

In modern digital life , a single user may have a lot of online accounts, including :

- Email
- Social media
- Banking apps
- Any apps…

Because humans cannot remember many unique, strong passwords, they often:

- Reuse the same weak password across different services
- Store passwords in unsafe locations (notes apps, text files ,…)
- Forget login credentials
- Get exposed to credential-suffering attacks

This behavior leads to serious cybersecurity risk such as:

- Account hacked
- Data theft
- Unauthorized access

Therefore, users need a secure way to store their passwords that does not expose them to attackers.

1.3 Solution Overview

This project solve the problem by creating a locally encrypted password vault , protected using :

- Bcrypt hashing :Used to securely store the master password. Even if the vault file is stolen, attackers cannot recover the master password.
- PBKDF2 (Password-Based key Derivation Function):Used to convert the master password into a strong, 256-bit AES key through key stretching.
- AES Encryption (AES-GCM or AES-CBC): Used to encrypt each saved password entry. Only the correct master password can decrypt the data.
- JSON vault storage: All encrypted entries are stored in an easy-to-read but secure JSON file.

This ensures that:

- Passwords are never stored in plaintext
- Even the developer cannot see user passwords
- Attackers cannot decrypt the vault without the correct master password

1.4 Motivation

This project is motivated by :

- Real world cybersecurity needs:

Password managers are a critical part of modern security. Understanding their design helps improve such as Authentication mechanisms, Key management, Encryption practices.

- Hand on cryptography learning :

Instead of study on Cryptography subject , this project will show the implementation of what I learned like about how encryption protects real data, how hashing prevents password theft.

- Practical usefulness:

The project has direct value such as : Users can securely store credentials, Student learn how encrypted vaults work.

1.5 Related Cryptography Concepts

This project integrated multiple cryptographic mechanisms:

- Bcrypt : Slow, adaptive password hashing algorithm.
- Salt: Random value added before hashing.
- PBKDF2: Converts a password into a strong AES key.
- IV/Nonce: Ensures identical passwords do not produce identical ciphertexts.
- JSON Storage: Stores encrypted entries.

II. System Design / Architecture

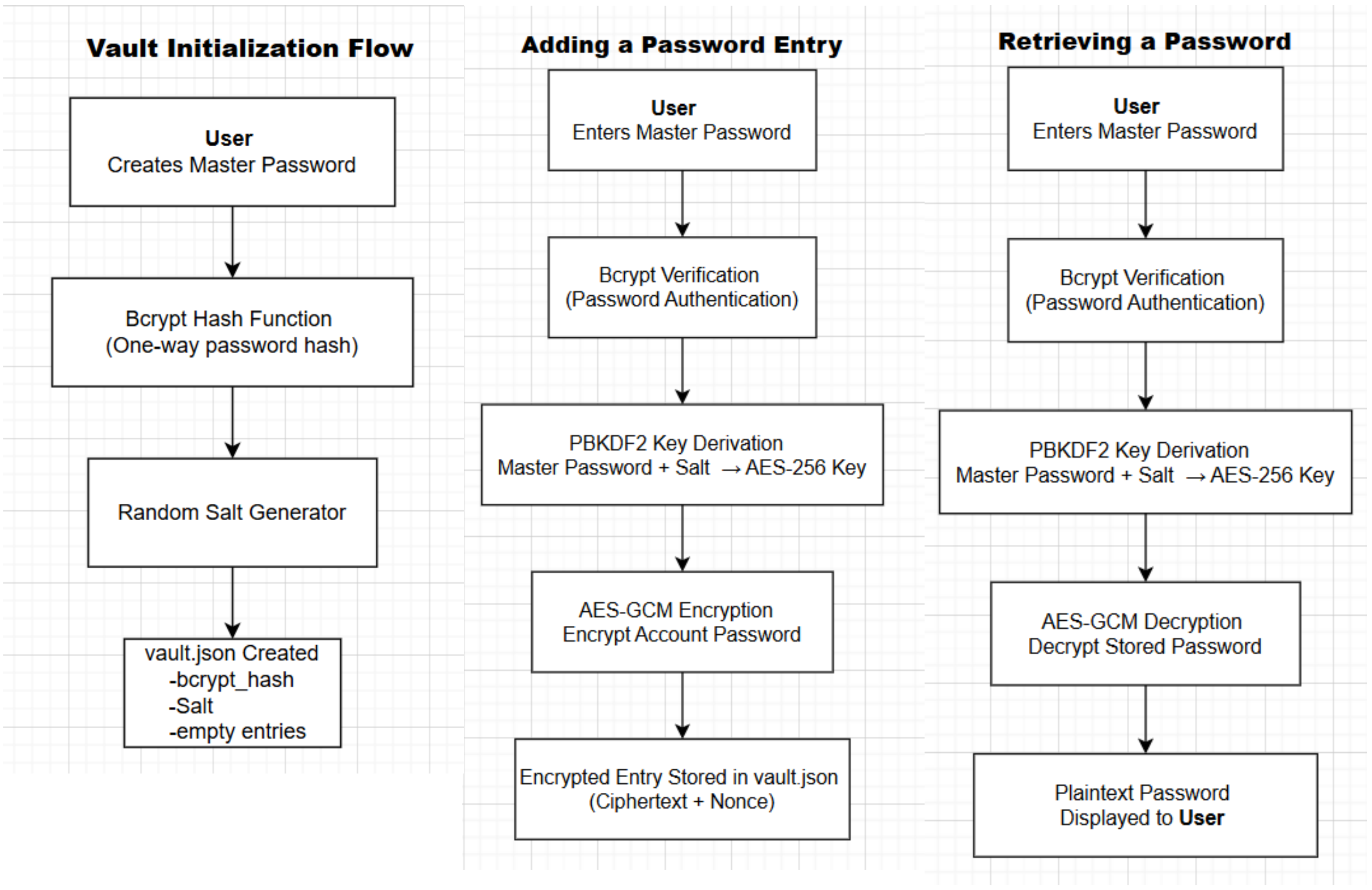
2.1 System overview

The system is designed as a command-line application with modular components. Each file in the project has a clear responsibility:

- main.go : Handleuser commands and program flow
- crypto.go : implements cryptographic operations
- strorage.go : handles file storage and encoding
- vault.go : defines data structures

All components belong to the same Go package (main) and work together to provide secure password management.

2.2 Data flow / Encrypt process



III. Implementation Details

3.1 Data structures

Vault Structure

The vault structure stores vault metadata and encrypted entires:

- Bcrypthash : Hashed master password
- salt : Based64-encoded salt
- entries : list of encrypted credentials

Entry Structure

Each entry contains:

- Site name

- Username
- Encrypted password
- Encryption nonce
- timestamp

3.2 Cryptographic functions

- HashMasterPassword()
  Uses bcrypt to securely hash the master password.
- VerifyMasterPassword()
  Verifies password input against stored bcrypt hash.
- DeriveKey()
  Uses PBKDF2 with SHA-256 to generate a strong AES key.
- EncryptAES()
  Encrypts passwords using AES-GCM.
- DecryptAES()
  Decrypts stored passwords after authentication.

3.3Security Design considerations

- Password are never stored in plaintext
- Vault file is useless without master password
- High PBKDF iteration count resists brute-force attacks
- AES ensures confidentiality and integrity

IV. Usage Guide

4.1 Requirements

- Go version 1.22 or later
- Golang.org/x/crypto package

4.2 Build and Run

Installation & Setup

Follow these steps exactly to avoid errors.

- Step 1: Verify Go Installation

```
go version
```

Output should show Go 1.22 or later.

- Step 2: Enable Go Modules (Important on Windows)

```
go env GO111MODULE
```

If it is off, enable it:

```
setx GO111MODULE on
```

Restart the terminal after running this command.

- Step 3: Install Dependencies

From the project root directory:

```
go mod tidy
```

This command:

  - Downloads golang.org/x/crypto
  - Fixes missing or unused dependencies
  - Generates go.sum

- Step 4: Build the Project (Optional but Recommended)

```
go build .
```

If no error appears, the environment is correctly set up.

Running the Program

Important: Do NOT run only main.go. This project consists of multiple files that must be compiled together.

Correct way:

```
go run .
```

or with a command:

```
go run . init

go run . add

go run . list

go run . get Gmail
```

Incorrect (causes undefined function errors):

```
go run main.go
```

Vault File Requirement

- The file vault.json is automatically created when running:

go run . init

- Users should not create or edit this file manually
- All passwords inside are encrypted

4.3 Example Output

```
Master password:

Site: Gmail

Username: user@gmail.com

Password: ********

Entry saved!
```

Usage Examples

1. Initialize the Vault

Creates a new encrypted vault and master password.

```
go run . init
```

2. Add a Password Entry

Stores a new encrypted credential.

```
go run . add
```

3. List All Stored Entries

Displays all stored passwords after authentication.

```
go run . list
```

4. Retrieve a Specific Entry

Gets the password for a specific site.

```
go run . get <site>
```

Example:

```
go run . get Gmail
```

go mod tidy
go run . init
go run . add
go run . list
go run . get <site>

4.4 Repository / Version control

- Only the following files are pushed to GitHub:

File	Reason
main.go	Source code
crypto.go	Cryptographic logic
vault.go	Data structures

storage.go	File handling
go.mod	Dependency definition
go.sum	Dependency integrity
README.md	Documentation
.gitignore	Professional practice

- The following files are not pushed:
    - secure-password-manager.exe TO compiled binary
    - vault.json → contains encrypted user data
    - Temporary logs (*.log)
- **.gitignore** is configured to prevent accidental commits of sensitive or unnecessary files:

```
*.exe
vault.json
*.log
```

This demonstrates secure version-control practices and professionalism.

V. Conclusion and Future work

5.1 Conclusion

This project successfully demonstrates how cryptography can be applied to protect sensitive user data. By combining hashing , PBKDF2 key derivation, and AES encryption, the system ensures that stored credentials remain secure even if the vault file is exposed.

The project meets its educational goal by transforming theoretical cryptography concepts into a practical, working system.

5.2Future work

Possible enhancements include:

- Master password change functionality
- Password generator
- Entry update and detection
- Graphical user interface
- Cloud-based encrypted backup

VI. Reference

Go Documentation – Official Go language documentation covering modules, standard library,and package usage.
https://go.dev/doc/

golang.org/x/crypto Library Documentation – Reference for Go cryptographic packages used in the project.
https://pkg.go.dev/golang.org/x/crypto