**Vault Of Codes: Mini Project**

Project Name: “ Secret Code Generator”

Mobile Num : +91 9345834909

Gmail : [prabhaprabhas039@gmail.com](mailto:prabhaprabhas039@gmail.com)

**Source Code:**

#!/usr/bin/env python3

"""

Secret Code Generator

- Save as: secret\_code\_generator.py

- Run: python secret\_code\_generator.py

Features:

- Generate random secrets (passwords / codes / passphrases)

- Encode/decode using:

    \* Base64

    \* Caesar cipher (simple shift)

    \* Vigenère cipher (keyword-based)

- Hashing (SHA-256) for one-way secret storage

- Save / load secrets to a local JSON "vault"

- Simple CLI menu

"""

import base64

import json

import os

import random

import secrets

import string

import sys

from hashlib import sha256

from typing import Tuple

VAULT\_FILE = "secret\_vault.json"

# -------------------------

# Utility helpers

# -------------------------

def ensure\_vault():

    if not os.path.exists(VAULT\_FILE):

        with open(VAULT\_FILE, "w") as f:

            json.dump({"items": []}, f)

def save\_to\_vault(name: str, method: str, value: str):

    ensure\_vault()

    with open(VAULT\_FILE, "r+") as f:

        data = json.load(f)

        data["items"].append({"name": name, "method": method, "value": value})

        f.seek(0)

        json.dump(data, f, indent=2)

        f.truncate()

def list\_vault():

    ensure\_vault()

    with open(VAULT\_FILE, "r") as f:

        data = json.load(f)

    return data.get("items", [])

# -------------------------

# Secret generation

# -------------------------

def generate\_password(length: int = 16, use\_symbols: bool = True,

                      use\_digits: bool = True, use\_upper: bool = True,

                      use\_lower: bool = True) -> str:

    if length <= 0:

        raise ValueError("length must be > 0")

    pool = ""

    if use\_lower:

        pool += string.ascii\_lowercase

    if use\_upper:

        pool += string.ascii\_uppercase

    if use\_digits:

        pool += string.digits

    if use\_symbols:

        # A safe set of symbols for passwords

        pool += "!@#$%^&\*()-\_=+[]{};:,.<>/?"

    if not pool:

        raise ValueError("No character sets selected.")

    # Use secrets.choice for cryptographic randomness

    return "".join(secrets.choice(pool) for \_ in range(length))

def generate\_passphrase(words: int = 4, separator: str = "-") -> str:

    # Small built-in wordlist; for production, use a larger list or wordlist file

    WORDLIST = [

        "coffee", "candle", "rocket", "marble", "ocean", "forest", "laptop",

        "river", "mountain", "planet", "keyboard", "mouse", "window", "lamp",

        "paper", "gold", "silent", "echo", "crystal", "shadow"

    ]

    return separator.join(secrets.choice(WORDLIST) for \_ in range(words))

# -------------------------

# Base64 encode / decode

# -------------------------

def base64\_encode(text: str) -> str:

    b = text.encode("utf-8")

    return base64.b64encode(b).decode("utf-8")

def base64\_decode(b64: str) -> str:

    return base64.b64decode(b64.encode("utf-8")).decode("utf-8")

# -------------------------

# Caesar cipher (shift)

# -------------------------

def caesar\_shift\_char(ch: str, shift: int) -> str:

    if ch.isalpha():

        base = 'A' if ch.isupper() else 'a'

        return chr((ord(ch) - ord(base) + shift) % 26 + ord(base))

    return ch

def caesar\_encrypt(plain: str, shift: int) -> str:

    return "".join(caesar\_shift\_char(ch, shift) for ch in plain)

def caesar\_decrypt(cipher: str, shift: int) -> str:

    return caesar\_encrypt(cipher, -shift)

# -------------------------

# Vigenère cipher

# -------------------------

def vigenere\_encrypt(plain: str, key: str) -> str:

    if not key:

        raise ValueError("Key must not be empty.")

    out = []

    key = [ord(k.lower()) - ord('a') for k in key if k.isalpha()]

    if not key:

        raise ValueError("Key must contain alphabetic characters.")

    ki = 0

    for ch in plain:

        if ch.isalpha():

            shift = key[ki % len(key)]

            base = 'A' if ch.isupper() else 'a'

            out.append(chr((ord(ch) - ord(base) + shift) % 26 + ord(base)))

            ki += 1

        else:

            out.append(ch)

    return "".join(out)

def vigenere\_decrypt(cipher: str, key: str) -> str:

    if not key:

        raise ValueError("Key must not be empty.")

    out = []

    key = [ord(k.lower()) - ord('a') for k in key if k.isalpha()]

    if not key:

        raise ValueError("Key must contain alphabetic characters.")

    ki = 0

    for ch in cipher:

        if ch.isalpha():

            shift = key[ki % len(key)]

            base = 'A' if ch.isupper() else 'a'

            out.append(chr((ord(ch) - ord(base) - shift) % 26 + ord(base)))

            ki += 1

        else:

            out.append(ch)

    return "".join(out)

# -------------------------

# Hashing

# -------------------------

def sha256\_hash(text: str) -> str:

    return sha256(text.encode("utf-8")).hexdigest()

# -------------------------

# Password strength (basic)

# -------------------------

def password\_strength(pwd: str) -> Tuple[int, str]:

    score = 0

    if len(pwd) >= 8:

        score += 1

    if any(c.islower() for c in pwd):

        score += 1

    if any(c.isupper() for c in pwd):

        score += 1

    if any(c.isdigit() for c in pwd):

        score += 1

    if any(c in "!@#$%^&\*()-\_=+[]{};:,.<>/?\\" for c in pwd):

        score += 1

    if score <= 2:

        return score, "Weak"

    elif score == 3:

        return score, "Moderate"

    elif score == 4:

        return score, "Strong"

    else:

        return score, "Very Strong"

# -------------------------

# CLI / Menu

# -------------------------

def print\_menu():

    print("\n=== Secret Code Generator ===")

    print("1) Generate random password")

    print("2) Generate passphrase")

    print("3) Base64 encode")

    print("4) Base64 decode")

    print("5) Caesar encrypt")

    print("6) Caesar decrypt")

    print("7) Vigenère encrypt")

    print("8) Vigenère decrypt")

    print("9) SHA-256 hash")

    print("10) Save last result to vault")

    print("11) List vault")

    print("0) Exit")

def main():

    last\_result = None

    while True:

        print\_menu()

        try:

            choice = input("Choose an option: ").strip()

        except (EOFError, KeyboardInterrupt):

            print("\nExiting.")

            return

        if choice == "1":

            length = input("Length (default 16): ").strip()

            length = int(length) if length else 16

            pwd = generate\_password(length=length)

            last\_result = pwd

            score, label = password\_strength(pwd)

            print(f"\nGenerated password: {pwd}")

            print(f"Strength: {label} (score {score}/5)")

        elif choice == "2":

            words = input("Number of words (default 4): ").strip()

            words = int(words) if words else 4

            sep = input("Separator (default '-'): ").strip() or "-"

            p = generate\_passphrase(words=words, separator=sep)

            last\_result = p

            print(f"\nGenerated passphrase: {p}")

        elif choice == "3":

            text = input("Text to encode: ")

            res = base64\_encode(text)

            last\_result = res

            print(f"\nBase64: {res}")

        elif choice == "4":

            text = input("Base64 to decode: ")

            try:

                res = base64\_decode(text)

                last\_result = res

                print(f"\nDecoded: {res}")

            except Exception as e:

                print("Invalid base64 string or decode error:", e)

        elif choice == "5":

            plain = input("Plain text: ")

            shift = input("Shift (integer, default 3): ").strip()

            shift = int(shift) if shift else 3

            res = caesar\_encrypt(plain, shift)

            last\_result = res

            print(f"\nCaesar encrypted: {res}")

        elif choice == "6":

            cipher = input("Cipher text: ")

            shift = input("Shift (integer, default 3): ").strip()

            shift = int(shift) if shift else 3

            res = caesar\_decrypt(cipher, shift)

            last\_result = res

            print(f"\nCaesar decrypted: {res}")

        elif choice == "7":

            plain = input("Plain text: ")

            key = input("Key (letters only): ").strip()

            try:

                res = vigenere\_encrypt(plain, key)

                last\_result = res

                print(f"\nVigenère encrypted: {res}")

            except Exception as e:

                print("Error:", e)

        elif choice == "8":

            cipher = input("Cipher text: ")

            key = input("Key (letters only): ").strip()

            try:

                res = vigenere\_decrypt(cipher, key)

                last\_result = res

                print(f"\nVigenère decrypted: {res}")

            except Exception as e:

                print("Error:", e)

        elif choice == "9":

            text = input("Text to hash: ")

            res = sha256\_hash(text)

            last\_result = res

            print(f"\nSHA-256: {res}")

        elif choice == "10":

            if last\_result is None:

                print("No 'last result' to save. Generate or encode something first.")

            else:

                name = input("Friendly name for this secret (e.g. 'gmail-password'): ").strip()

                method = input("Method (e.g. 'generated', 'base64', 'caesar', 'sha256'): ").strip() or "unknown"

                save\_to\_vault(name=name or "unnamed", method=method, value=last\_result)

                print("Saved to vault.")

        elif choice == "11":

            items = list\_vault()

            if not items:

                print("Vault is empty.")

            else:

                print("\nVault contents:")

                for i, it in enumerate(items, 1):

                    print(f"{i}. {it['name']} [{it['method']}] : {it['value']}")

        elif choice == "0":

            print("Goodbye.")

            return

        else:

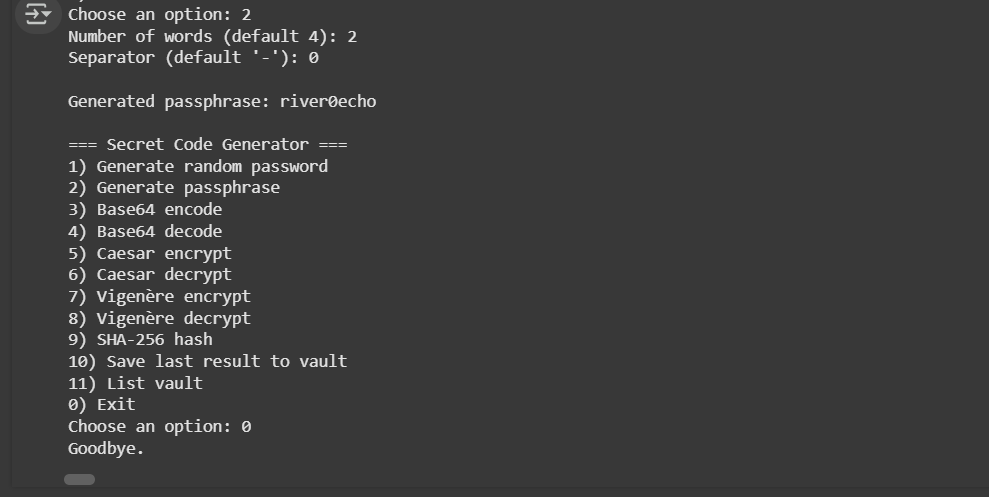
            print("Invalid choice. Please choose an option from the menu.")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output:**

****

****

P Prabhash

(Signature)