SOURCE CODE

```
import tkinter as tk
from tkinter import ttk, messagebox
import hashlib
import random
import string
from itertools import product
import threading
import re

# Load common passwords
def load_common_passwords():
    try:
        with open("common_passwords.txt", "r") as file:
            return [line.strip() for line in file]
    except FileNotFoundError:
        return ["password", "12345678", "qwertyui", "iloveyou", "letmein", "admin123", "welcome1", "password1"]

def generate_random_password(length=8):
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for _ in range(length))
    with open("generated_passwords.txt", "a") as file:
        file.write(password + "\n")
    return password

def generate_password():
    try:
        length = int(entry_length.get())
        if length < 4 or length > 16:
            messagebox.showwarning("Warning", "Choose a length between 4 and 16.")
            return
        password = generate_random_password(length)
        entry_message.delete(0, tk.END)
        entry_message.insert(0, password)
    except ValueError:
```

```
        messagebox.showwarning("Warning", "Please enter a valid number.")

def hash_message():
    message = entry_message.get()
    selected_algo = combo_algo.get()

    if not message:
        result_label.config(text="⚠ Please enter a message.")
        return

    algo_map = {
        "SHA-1": hashlib.sha1,
        "SHA-256": hashlib.sha256,
        "SHA-512": hashlib.sha512,
        "MD5": hashlib.md5
    }

    hash_func = algo_map.get(selected_algo)
    if not hash_func:
        result_label.config(text="⚠ Unsupported algorithm.")
        return

    global current_hash_value
    current_hash_value = hash_func(message.encode()).hexdigest()
    result_label.config(text=f"🔒 Hashed Value:\n{current_hash_value}")

def copy_to_clipboard():
    if current_hash_value:
        root.clipboard_clear()
        root.clipboard_append(current_hash_value)
        root.update()
        messagebox.showinfo("Copied", "✅ Hash copied to clipboard!")
    else:
        messagebox.showwarning("Warning", "⚠ No hash value to copy!")
```

```
def dictionary_attack(hash_value, hash_func):
    common_passwords = load_common_passwords()
    try:
        with open("generated_passwords.txt", "r") as file:
            common_passwords.extend([line.strip() for line in file])
    except FileNotFoundError:
        pass

    for password in common_passwords:
        if hash_func(password.encode()).hexdigest() == hash_value:
            return password
    return None

def start_progress():
    progress_bar.grid(row=9, column=0, columnspan=3, pady=(5, 0))
    progress_bar.start()

def stop_progress():
    progress_bar.stop()
    progress_bar.grid_forget()

def is_valid_hash(hash_value, algo):
    if not re.fullmatch(r'[a-fA-F0-9]+', hash_value):
        return False

    expected_lengths = {
        "SHA-1": 40,
        "SHA-256": 64,
        "SHA-512": 128,
        "MD5": 32
    }

    return len(hash_value) == expected_lengths.get(algo, 0)

def brute_force_crack():
```

```
hash_value = entry_hash.get().strip()
selected_algo = combo_algo.get()
characters = string.ascii_letters + string.digits + string.punctuation

if not hash_value:
    messagebox.showwarning("Warning", "⚠ Please enter a hash value to crack.")
    return

if not is_valid_hash(hash_value, selected_algo):
    messagebox.showerror("Invalid Hash", "⚠ Please enter a valid hash value.")
    return

algo_map = {
    "SHA-1": hashlib.sha1,
    "SHA-256": hashlib.sha256,
    "SHA-512": hashlib.sha512,
    "MD5": hashlib.md5
}

hash_func = algo_map.get(selected_algo)
if not hash_func:
    messagebox.showwarning("Warning", "⚠ Unsupported algorithm.")
    return

password = dictionary_attack(hash_value, hash_func)
if password:
    cracked_result_label.config(text=f"🔓 Cracked (Dictionary): {password}")
    stop_progress()
    return

def attempt_crack():
    start_progress()
    max_length = 5
    for length in range(1, max_length + 1):
        for attempt in product(characters, repeat=length):
```

```
            password = ''.join(attempt)
            if hash_func(password.encode()).hexdigest() == hash_value:
                cracked_result_label.config(text=f"🔓 Cracked (Brute-force): {password}")
                stop_progress()
                return
        cracked_result_label.config(text="❌ No match found.")
        stop_progress()

    threading.Thread(target=attempt_crack, daemon=True).start()

# === GUI ===
root = tk.Tk()
root.title("🔐 Password Hash Cracker")
root.configure(bg="#eef7ff")
root.geometry("700x500")

style = ttk.Style(root)
style.theme_use("clam")
style.configure("TFrame", background="#eef7ff")
style.configure("TLabel", background="#eef7ff", font=("Segoe UI", 10))
style.configure("TButton", font=("Segoe UI", 9, "bold"), padding=6)
style.configure("TProgressbar", thickness=15)

frame = ttk.Frame(root, padding="20 15")
frame.pack(expand=True, fill="both")

# Title
title = ttk.Label(frame, text="🔐 Password Hash Generator & Cracker", font=("Segoe UI", 16, "bold"),
foreground="#005b96")
title.grid(row=0, column=0, columnspan=3, pady=(0, 20))

# Generate Section
ttk.Label(frame, text="Password Length:").grid(row=1, column=0, sticky="e")
entry_length = ttk.Entry(frame, width=5, justify="center")
entry_length.insert(0, "8")
```

```
entry_length.grid(row=1, column=1)

ttk.Button(frame, text="Generate Random Password", command=generate_password).grid(row=1, column=2, padx=5)

entry_message = ttk.Entry(frame, width=50, justify="center")
entry_message.grid(row=2, column=0, columnspan=3, pady=10)

ttk.Label(frame, text="Select Hash Algorithm:").grid(row=3, column=0, sticky="e")
combo_algo = ttk.Combobox(frame, values=["SHA-1", "SHA-256", "SHA-512", "MD5"], width=12)
combo_algo.set("SHA-256")
combo_algo.grid(row=3, column=1, sticky="w")

ttk.Button(frame, text="Generate Hash", command=hash_message).grid(row=3, column=2, pady=10)

result_label = ttk.Label(frame, text="🔒 Hashed Value:", wraplength=600, justify="left")
result_label.grid(row=4, column=0, columnspan=3, pady=10)

ttk.Button(frame, text="Copy Hash", command=copy_to_clipboard).grid(row=5, column=1, pady=5)

# Crack Section
ttk.Separator(frame, orient="horizontal").grid(row=6, column=0, columnspan=3, sticky="ew", pady=10)
ttk.Label(frame, text="Enter Hash to Crack:").grid(row=7, column=0, sticky="e")

entry_hash = ttk.Entry(frame, width=50, justify="center")
entry_hash.grid(row=7, column=1, columnspan=2, pady=5)

ttk.Button(frame, text="Crack Hash", command=brute_force_crack).grid(row=8, column=1, pady=10)

progress_bar = ttk.Progressbar(frame, mode="indeterminate", length=300)
cracked_result_label = ttk.Label(frame, text="🔓 Cracked Password:", foreground="green")
cracked_result_label.grid(row=10, column=0, columnspan=3, pady=10)

# Run App
root.mainloop()
```