WORKING OF THIS CODE

```
import tkinter as tk
from tkinter import messagebox, scrolledtext
from cryptography.fernet import Fernet
import os
```

What I'm doing here:

I'm using tkinter to create a GUI (graphical interface) instead of just a boring terminal.
messagebox is for showing popup messages like success or errors.
scrolledtext is a special text area that lets users scroll if the text is long.
Fernet is the tool from the cryptography library which handles encryption and decryption securely.
os helps me check whether the encryption key file already exists or not.

Generating the Key

```
def generate_key():
    key = Fernet.generate_key()
    with open("key.key", "wb") as key_file:
        key_file.write(key)
    messagebox.showinfo("Success", "Key generated and saved as key.key")
```

Explanation:

This function creates a secure encryption key and saves it to a file named key.key.
The user just has to click a button, and the key is auto-generated.
I also added a pop-up that says the key was successfully created.

Loading the Key

```
def load_key():
    if not os.path.exists("key.key"):
        messagebox.showerror("Error", "Key not found! Please generate it first.")
        return None
    with open("key.key", "rb") as key_file:
        return key_file.read
```

Explanation:

Before encrypting or decrypting, I need to use the key.
This function checks: Is the key file already there?
If not, I show an error message.
If yes, I open and read the key to use it.

Encrypting the Message:

```
def encrypt_message():
    key = load_key()
    if not key:
        return
    f = Fernet(key)
    message = input_text.get("1.0", tk.END).strip()
    if not message:
    messagebox.showwarning("Warning", "Input message is empty!")
        return
    encrypted = f.encrypt(message.encode())
    output_text.config(state=tk.NORMAL)
    output_text.delete("1.0", tk.END)
    output_text.insert(tk.END, encrypted.decode())
    output_text.config(state=tk.DISABLED)
```

Explanation:
This is where the magic happens – encryption!
First, I load the key.
I use the Fernet object with the key to prepare for encryption.
Then, I grab the message the user typed in the input box.
If the input is empty, I warn the user.
Otherwise, I encrypt it and display the encrypted version in the output box.

Decrypting the Message:

```
def decrypt_message():
    key = load_key()
    if not key:
        return
    f = Fernet(key)
    encrypted_message = input_text.get("1.0", tk.END).strip()
    if not encrypted_message:
        messagebox.showwarning("Warning", "Encrypted message is empty!")
        return
    try:
        decrypted = f.decrypt(encrypted_message.encode())
        output_text.config(state=tk.NORMAL)
        output_text.delete("1.0", tk.END)
        output_text.insert(tk.END, decrypted.decode())
      output_text.config(state=tk.DISABLED)
    except Exception:
        messagebox.showerror("Error", "Invalid encrypted message or wrong key!")
```

Explanation:
This does the opposite of encryption — it decrypts the message.
I load the key and read the encrypted text from the input box.

If it’s empty, I show a warning.
Then, I try to decrypt it. If it's successful, I show the original message.
If something goes wrong (maybe wrong key or bad input), I show an error.

Setting up the Main Window (GUI):

```
root = tk.Tk()
root.title("Encryption-Decryption Tool")
root.geometry("700x500")
root.configure(bg="#f7f7f7")
```

Explanation:
This part creates the main window of the app.
I gave it a title and a fixed size.
The background color is soft light grey to make it look clean.

Fonts and Button Styles:

```
FONT_TITLE = ("Segoe UI", 14, "bold")
FONT_TEXT = ("Segoe UI", 11)
BTN_STYLE = {
    "font": ("Segoe UI", 11),
    "width": 16,
    "padx": 5,
    "pady": 5,
    "bg": "#74b9ff",
    "fg": "#2d3436"
}
```

Explanation:
I want the UI to look modern and readable.
So I defined styles for titles, text areas, and buttons using nice fonts and colors.

Input Label and Text Area:

```
tk.Label(root, text="Enter your message or encrypted text", font=FONT_TITLE,
fg="#2d3436", bg="#f7f7f7").pack(pady=10)
input_text = scrolledtext.ScrolledText(root, width=70, height=6, font=FONT_TEXT,
bg="white", fg="#2d3436", insertbackground="#2d3436")
input_text.pack(padx=20, pady=5)
```

Explanation:
I added a label to guide the user.
Then a scrollable text box where users can type in messages or paste encrypted ones.

Buttons for Encrypt, Decrypt, Generate Key:

```
btn_frame = tk.Frame(root, bg="#f7f7f7")
btn_frame.pack(pady=15)

encrypt_btn = tk.Button(btn_frame, text="🔒 Encrypt", command=encrypt_message,
**BTN_STYLE)
encrypt_btn.grid(row=0, column=0, padx=10)

decrypt_btn = tk.Button(btn_frame, text="🔓 Decrypt", command=decrypt_message,
**BTN_STYLE)
decrypt_btn.grid(row=0, column=1, padx=10)

key_btn = tk.Button(btn_frame, text="🗝️ Generate Key", command=generate_key,
**BTN_STYLE)
key_btn.grid(row=0, column=2, padx=10)
```

Explanation:
Here, I made a nice row of buttons:
Encrypt: Encrypts whatever is written.
Decrypt: Decrypts the given encrypted message.
Generate Key: Creates the key used for both encryption and decryption.

Output Label and Text Box:

```
tk.Label(root, text="Output", font=FONT_TITLE, fg="#2d3436", bg="#f7f7f7").pack(pady=10)
output_text = scrolledtext.ScrolledText(root, width=70, height=6, font=FONT_TEXT,
bg="white", fg="#2d3436", state=tk.DISABLED)
output_text.pack(padx=20, pady=5)
```

Explanation:
I added a label "Output" and another scrollable text box for showing results.
This one is read-only, so users can't accidentally edit the output.

Finally, Start the App:

```
root.mainloop()
```

Explanation:
This line keeps the app window running so the user can interact with it.
It's like saying: "Okay, launch the app and wait for user actions!"

Summary of project:

This is a simple yet powerful encryption and decryption tool that I built using Python's tkinter library for the user interface and the cryptography library for securely handling sensitive messages. The idea behind this tool is to allow the user to either encrypt a plain message or decrypt an encrypted one, all through an easy-to-use GUI.