The code includes:

A tkinter GUI with sections for hiding and extracting.

1. File dialogs (tkinter.filedialog) to select images and save output.
2. cv2 (opencv-python) to read, manipulate pixel data (LSB modification), and write images.
3. os for basic path validation.
4. A simple text entry for the secret message.
5. A text area to display the extracted message.
6. Basic error handling and user feedback using tkinter.messagebox.

**Prerequisites:**

You need to install the required libraries:

Bash: - pip install opencv-python numpy

(numpy is automatically used by OpenCV).

**Cryptography technique:**

**1. Install the cryptography library:**

Bash: - pip install cryptography

**2. Conceptual Changes:**

* **Encoding:**
  1. Get the secret message (text) and the password from the user.
  2. **Encrypt** the message using the password. This involves:
     + Deriving a strong encryption key from the password using a Key Derivation Function (KDF) like PBKDF2. This requires a salt (random data stored with the ciphertext).
     + Using an authenticated encryption mode like AES-GCM, which requires a nonce (number used once, also stored).
     + The output will be salt + nonce + ciphertext + authentication\_tag.
  3. Convert this combined **encrypted byte data** into a binary string.
  4. Append the DELIMITER\_BIN.
  5. Embed this final binary string into the image using LSB.
* **Decoding:**
  1. Extract the full binary string using LSB until the DELIMITER\_BIN is found.
  2. Remove the delimiter bits.
  3. Convert the remaining binary string back into **bytes** (this is the salt + nonce + ciphertext + tag).
  4. Get the password from the user.
  5. **Decrypt** the data:
     + Extract the salt and nonce from the beginning of the byte data.
     + Derive the decryption key from the password and the *extracted salt* using PBKDF2 (with the same parameters).
     + Use AES-GCM with the derived key and *extracted nonce* to decrypt the remaining ciphertext+tag. The library automatically verifies the authenticity tag.
     + If the password is wrong, the salt/nonce is wrong, or the data was tampered with, decryption will fail.
  6. If successful, convert the decrypted bytes back into a UTF-8 text string.
  7. Display the result.

**Explanation:**

1. **Imports:** Imports necessary modules (tkinter, ttk, filedialog, messagebox, scrolledtext, cv2, os, numpy, logging).
2. **Configuration:** Defines a DELIMITER string. This is crucial for the decoding process to know when the hidden message ends. Its binary representation is pre-calculated. logging is set up for better debugging info in the console.
3. **text\_to\_binary/binary\_to\_text:** Helper functions to convert text to a stream of '0's and '1's and back, using UTF-8 encoding.
4. **encode\_lsb Function:**
   * Reads the cover image using cv2.imread.
   * Checks if the image was read successfully.
   * Calculates the maximum data that can be hidden.
   * Appends the DELIMITER to the secret message.
   * Converts the message+delimiter to binary.
   * Checks if the binary message fits within the image's capacity.
   * Iterates through the image pixels (and their color channels B, G, R).
   * For each pixel channel value, it modifies the Least Significant Bit (LSB) to match the corresponding bit from the binary message.
   * It stops once all message bits are embedded.
   * Saves the modified image using cv2.imwrite. **Important:** Saving as PNG or BMP (lossless formats) is recommended. JPEG is lossy and will corrupt the hidden data.
   * Includes error handling (try...except) for file issues, size errors, etc.
5. **decode\_lsb Function:**
   * Reads the stego image.
   * Iterates through pixels/channels, extracting the LSB from each value.
   * Appends the extracted bits ('0' or '1') to a string (extracted\_binary).
   * Continuously checks if the end of the extracted\_binary string matches the pre-calculated DELIMITER\_BIN.
   * Once the delimiter is found, it stops extracting.
   * It takes the binary string *before* the delimiter bits.
   * Converts this binary data back to text using binary\_to\_text.
   * Includes error handling.
6. **StegoApp Class (Tkinter GUI):**
   * Inherits from tk.Tk to create the main window.
   * Sets up window title and size.
   * Uses ttk.Notebook to create separate tabs for "Hide" and "Extract".
   * Defines tk.StringVar variables to easily link entry fields to Python variables.
   * **setup\_encode\_tab/setup\_decode\_tab:** Create labels, entry fields, buttons, and the scrolled text areas within each tab using ttk widgets and the grid layout manager.
   * **select\_file/save\_file:** Methods using filedialog to handle Browse for files and specifying save locations.
   * **run\_encode/run\_decode:** These methods are called when the main buttons are clicked. They retrieve input values from the GUI widgets, call the appropriate core LSB function (encode\_lsb or decode\_lsb), and display results or errors using messagebox and update the text area. They also perform basic input validation.
7. **if \_\_name\_\_ == "\_\_main\_\_"::** Standard Python practice to create an instance of the StegoApp and start the tkinter event loop (app.mainloop()) when the script is run directly.