SAMIR WALID SAMIR

412200040

SamirCryptoTool Documentation

Welcome to the documentation for Samir's Crypto Toolkit, a user-friendly app for exploring cryptography! This tool lets you encrypt, decrypt, hash, and sign messages using different techniques. It's built with Python and Tkinter, making it easy to use with a graphical interface. This document explains each part of the code, what it does, and how it all fits together.

You can access the code from my GitHub:

[crypto_tool/tool.py at main · samir22418/crypto_tool (github.com)]

---

1. Imports and Setup

What It Does?

This section brings in all the tools (libraries) we need to make the app work. Think of it as gathering ingredients before cooking!

Explanation

- **Tkinter (tkinter, filedialog, messagebox, ttk): Used to create the app's window, buttons, text boxes, and tabs. It's like the canvas and paint for our app's look.**
- **Crypto Libraries (Crypto.Cipher, Crypto.Random, Crypto.Util): Provide AES and DES encryption methods, random key generation, and padding for messages.**
- **Cryptography Libraries (cryptography.hazmat): Handle RSA encryption, digital signatures, and hashing (like SHA-256).**
- **Other (base64, hashlib): Help with encoding data (base64) and creating hashes (MD5, SHA-1, etc.).**

2. Classic Ciphers

What It Does

This part defines two simple, old-school ciphers (Caesar and Vigenere) for the "Classical" tab. They're fun ways to scramble messages using basic rules.

Explanation

- Caesar Cipher (caesar_cipher):
  - Takes a message (text) and shifts each letter by a number of spots (shift=3 by default).
  - Example: "A" becomes "D" (shifted 3 spots). Non-letters (like spaces) stay the same.
  - Uses ord() and chr() to convert letters to numbers and back, keeping uppercase/lowercase intact.
- Vigenere Cipher (vigenere_cipher):
  - Uses a keyword (default "crypto") to shift letters by different amounts.
  - Each letter in the message is shifted based on the corresponding letter in the keyword (e.g., "c" in "crypto" shifts by 2).
  - Loops through the keyword as needed and preserves non-letters.
- These functions are used in the "Classical" tab to let users play with historical ciphers.

3. Main App Class (SamirCryptoTool)

What It Does ?

This is the heart of the app! It sets up the window, creates the tabs, and prepares everything for the user to start exploring cryptography.

Explanation

- **Window Setup:**
  - **Creates a window (self.window) with the title "Samir's Crypto Toolkit" .**
  - **Sets the size to 700x900 pixels (narrower and taller, as per your edited image).**
  - **Uses a dark background (#1e1e2e) and allows resizing (resizable(True, True)).**
- **Theme (setup_look_and_feel):**
  - **Called to set up a dark, sleek theme (explained in the next section).**
- **Scrollable Area:**
  - **Adds a tk.Canvas and ttk.Scrollbar so users can scroll if content doesn't fit.**
  - **The main_area is a frame inside the canvas where all the tabs and other elements go.**
- **Title:**
  - **Displays "Samir's Crypto Toolkit" at the top in a bold, turquoise font (#5eead4).**
- **Tabs:**
  - **Uses ttk.Notebook to create tabs (like Symmetric, Asymmetric, etc.), set up by setup_tabs().**
- **Status Bar:**
  - **A small bar at the bottom (self.status) shows messages like "Ready to go!" or "Message signed!".**
- **Keys:**
  - **Initializes variables (self.aes_key, self.des_key, etc.) to store encryption keys for later use.**

4. Look and Feel (setup_look_and_feel)

What It Does

This part makes the app look nice with a dark theme, matching colors, and a consistent style for buttons, tabs, and text.

Explanation

- **Creates a custom theme called "crypto" with a dark color scheme:**
  - **Backgrounds: Mostly dark gray (#27272a for frames/tabs, #3f3f46 for text boxes).**
  - **Text Colors: Light gray (#d4d4d8) for labels, white (#ffffff) for text boxes and dropdowns.**
  - **Buttons: Blue (#3b82f6, darker #2563eb when clicked) with white text.**
  - **Tabs: Selected tabs turn turquoise (#5eead4) for visibility.**
- **Uses the "Montserrat" font for a modern look and "Consolas" for text boxes (like a code editor).**
- **This makes the app visually appealing and easy to read.**

---

5. Tab Creation (create_tab and setup_tabs)

What It Does

These functions set up the different sections (tabs) of the app, like Symmetric, Asymmetric, and Digital Signature. Each tab has a similar layout with text boxes, buttons, and options.

Explanation

- **Generic Tab Creator (create_tab):**
    - **Builds a tab with a standard layout: input box, key area (if needed), buttons, dropdown, and output box.**
    - **Input: A text box labeled "Message" (or custom label) where users type their message.**
    - **Keys: Adds a key input area:**
        - **For Symmetric: A single line for AES/DES keys (hidden with * for security) and a "Make New Key" button.**
        - **For Asymmetric/Digital Signature: Two text boxes for public and private RSA keys, plus a "Make RSA Keys" button.**
    - **Buttons: "Load File" to upload a text file, "Clear" to erase the input.**
    - **Dropdown: A menu to pick an action (e.g., "AES Encrypt", "Sign").**
    - **Action Button: A "Go!" button (or custom text like "Hash It!") to run the selected action.**
    - **Output: A "Result" text box to show the output, with a "Save Result" button.**
    - **Returns a dictionary with references to the widgets (e.g., input, output, option) for later use.**

- **Tab Setup (setup_tabs):**
    - **Creates five tabs:**

1. **Symmetric: For AES/DES encryption/decryption.**
2. **Asymmetric (RSA): For RSA encryption/decryption.**
3. **Classical: For Caesar and Vigenere ciphers (no keys needed).**
4. **Hashing: For creating hashes (no keys needed).**
5. **Digital Signature: For signing/verifying messages**
    - **Each tab is created by calling create_tab with specific settings (e.g., options, whether it needs keys).**

6. Key Generation

What It Does

These functions create keys for encryption and signing, so users don't have to make their own (which can be tricky!).

Explanation

- Symmetric Keys (generate_symmetric_key):
  - Checks the selected option in the Symmetric tab (e.g., "AES Encrypt").
  - For AES: Creates a 16-byte key (128 bits).
  - For DES: Creates an 8-byte key (64 bits).
  - Encodes the key in base64 (to make it text) and puts it in the key field.
  - Updates the status bar (e.g., "New AES key ready!").
- RSA Keys (generate_rsa_keys):
  - Creates a 2048-bit RSA key pair (private and public keys).
  - Converts the keys to text format (PEM) so they can be displayed.
  - Fills the public and private key fields in both the Asymmetric and Digital Signature tabs.
  - Updates the status bar with "New RSA keys created!".

---

7. Crypto Actions

What It Does

These functions handle the actual cryptography tasks, like encrypting, decrypting, hashing, and signing messages, based on the user's selections in each tab.

Explanation

- **Symmetric (run_symmetric):**
  - **Handles AES and DES encryption/decryption.**
  - **Checks if a key is provided and if it's the right size (16/24/32 bytes for AES, 8 bytes for DES).**
  - **Encrypts: Pads the message, encrypts it, and converts to hex.**
  - **Decrypts: Converts from hex, decrypts, and removes padding.**
  - **Shows the result in the output box or an error message if something goes wrong.**
- **Asymmetric (run_rsa):**
  - **Handles RSA encryption/decryption.**
  - **Encrypts with the public key, decrypts with the private key.**
  - **Uses OAEP padding for security and base64 encoding for the output.**
  - **Displays the result or an error if keys are missing.**
- **Classical (run_classic):**
  - **Applies the Caesar or Vigenere cipher to the message, using the functions defined earlier.**
  - **Simple and straightforward, with error handling for edge cases.**
- **Hashing (run_hashing):**
  - **Creates a hash (MD5, SHA-1, SHA-256, or SHA-512) of the message.**
  - **Converts the message to bytes, hashes it, and shows the result in hex.**
  - **Ensures the message isn't empty before proceeding.**
- **Digital Signature (run_digital_signature):**
  - **Sign: Uses the private key to sign the message with RSA-PSS and SHA-256, outputs the signature in base64.**
  - **Verify: Uses the public key to check if the signature matches the message. Shows "Signature checks out!" if valid, or "Signature doesn't match!" if not.**
  - **Requires both a key and (for verification) a signature to proceed.**

8. File Handling

What It Does

These functions let users load messages from a file or save their results to a file, making the app more practical.

Explanation

- **Load File (load_file):**
  - **Opens a file picker dialog where users can choose a .txt file (or any file).**
  - **Reads the file and puts its contents into the input text box.**
  - **Shows the file name in the status bar or an error if something goes wrong.**
- **Save Output (save_output):**
  - **Opens a save dialog where users can choose where to save their result.**
  - **Writes the output text to the file and shows a success message or error.**
- **Select All (select_all):**
  - **Adds a handy Ctrl+A shortcut to select all text in a text box, making it easier to copy or clear.**

---

9. Launch the App

What It Does

This final part starts the app, creating the window and running the main loop to keep it open.

Code

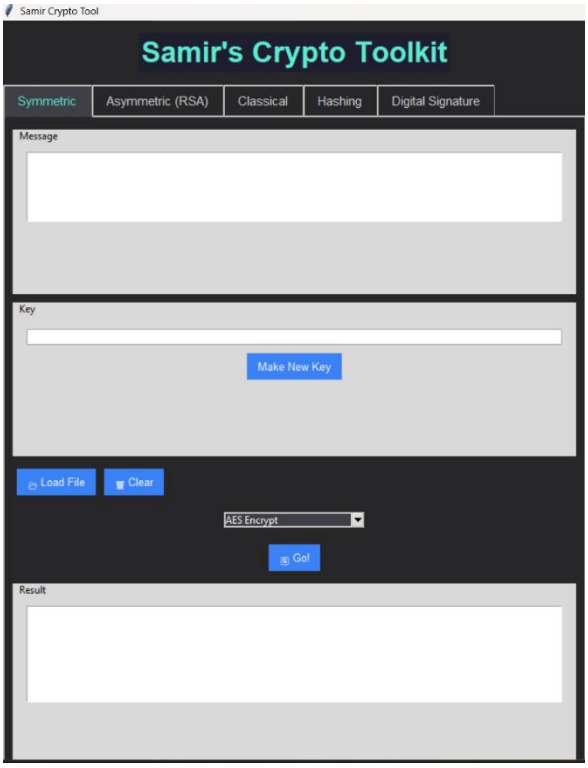
Explanation

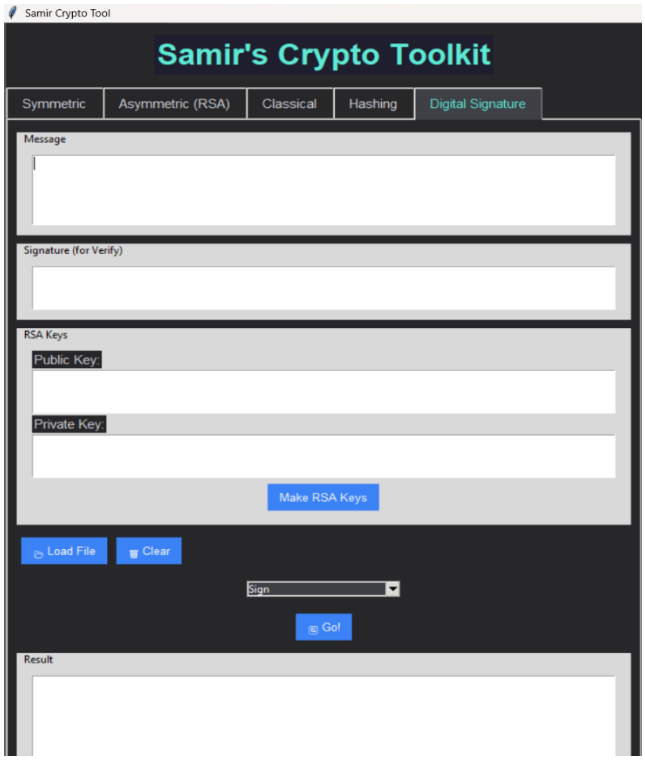
- **Creates a Tkinter window (tk.Tk()).**
- **Initializes the SamirCryptoTool app with the window.**
- **Starts the app's main loop (mainloop()), which keeps the window open and responsive to user actions.**

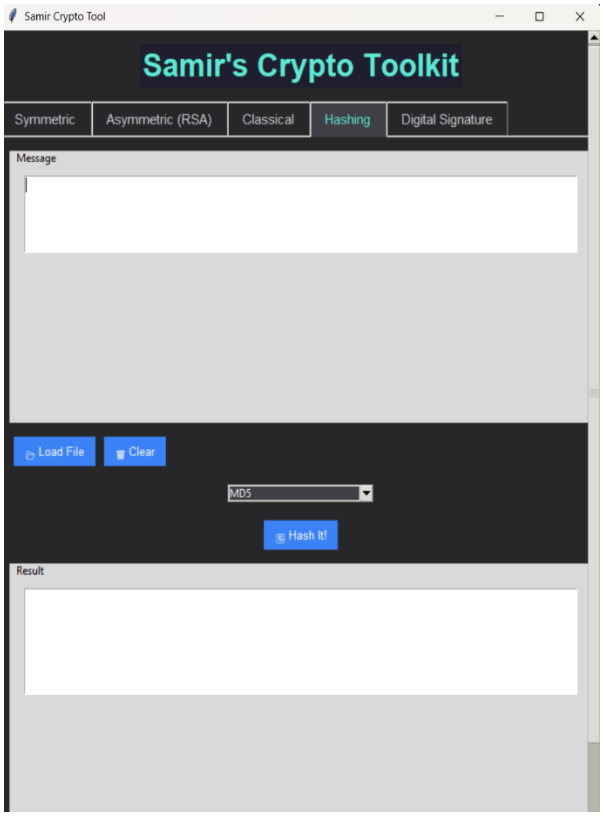
How to Use the App

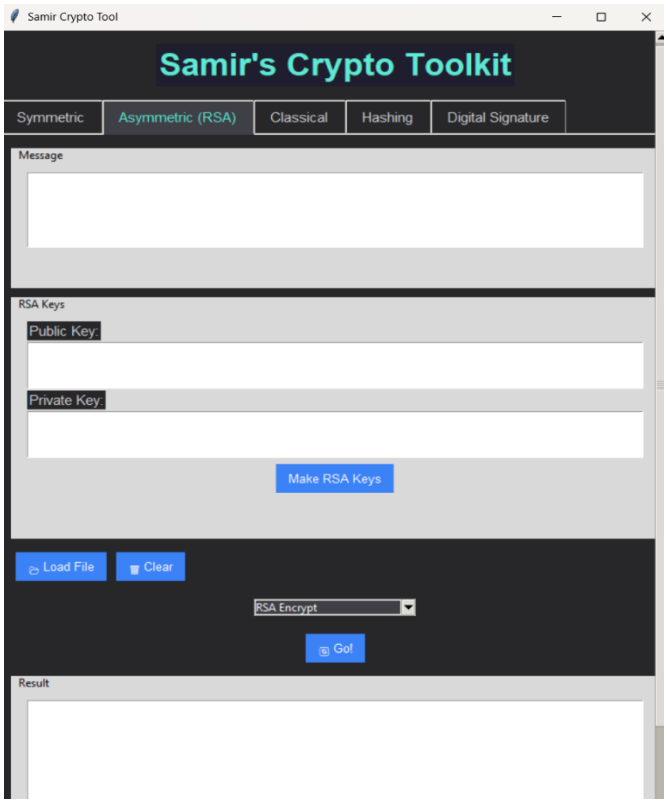
1. Run the Code:
    - Save the code in a file (e.g., samir_crypto_tool.py).
    - Make sure you have the required libraries installed (tkinter, pycryptodome, cryptography).
    - Run the script with Python: python samir_crypto_tool.py.
2. Explore the Tabs:
    - Symmetric: Encrypt/decrypt with AES or DES. Generate a key, type a message, and click "Go!".
    - Asymmetric (RSA): Encrypt with a public key, decrypt with a private key.
    - Classical: Try Caesar or Vigenere ciphers on your message.
    - Hashing: Create a hash (like SHA-256) of your message.
    - Digital Signature: Sign a message with a private key, or verify a signature with a public key.
3. Features:
    - Load a message from a file with "Load File".
    - Save your result with "Save Result".
    - Use Ctrl+A to select all text in a text box.
    - Scroll if the window is too small to show everything.

This documentation covers everything you need to understand and use Samir's Crypto Toolkit. Happy coding and exploring cryptography!


Samir Crypto Tool
Samir's Crypto Toolkit
Symmetric
Asymmetric (RSA)
Classical
Hashing
Digital Signature
Message
Key
Make New Key
Load File
Clear
AES Encrypt
Go!
Result


Samir Crypto Tool
Samir's Crypto Toolkit
Symmetric
Asymmetric (RSA)
Classical
Hashing
Digital Signature
Message
Signature (for Verify)
RSA Keys
Public Key:
Private Key:
Make RSA Keys
Load File
Clear
Sign
Go!
Result


Samir Crypto Tool
Samir's Crypto Toolkit
Symmetric
Asymmetric (RSA)
Classical
Hashing
Digital Signature
Message
Load File
Clear
MD5
Hash It!
Result


Samir Crypto Tool
Samir's Crypto Toolkit
Symmetric
Asymmetric (RSA)
Classical
Hashing
Digital Signature
Message
RSA Keys
Public Key:
Private Key:
Make RSA Keys
Load File
Clear
RSA Encrypt
Go!
Result