Project: BlueCrypt - Adaptive Hybrid Steganography Tool

Tagline: High-Capacity Secure Image Steganography using 2:2:4 Channel Optimization

Part 1: Project Implementation Details

1. Overview

BlueCrypt bridges the gap between basic student projects and expert-level security tools. Unlike traditional steganography that hides data sequentially (1 bit per pixel), BlueCrypt uses a **Hybrid Architecture** combining AES-256 Encryption with an **Adaptive LSB** strategy that favors the Blue channel of an image. This results in higher payload capacity and superior visual quality (PSNR > 55dB). To maximize presentation value and ease of use, the project utilizes a **Web Application Interface** powered by Python and Streamlit.

2. Implementation Plan (Streamlit Web App)

We will build this using **Python** due to its robust library support for image processing (Pillow), cryptography (PyCryptodome), and rapid web UI development (Streamlit).

Phase 1: The Cryptography Module

- **SHA-256 Hashing:** To convert user passwords into valid 32-byte encryption keys.
- **AES-GCM Encryption:** To encrypt the payload before embedding.
    - *Why:* The "Confidentiality Paradox"—we must encrypt before hiding so statistical analysis (like Chi-Square) sees random high-entropy noise rather than predictable text patterns.

Phase 2: The Steganography Engine (The Core)

- **2:2:4 Embedding Strategy:** (Based on Singh & Singh, 2015)
    - **Red Channel:** Embed 2 bits.
    - **Green Channel:** Embed 2 bits.
    - **Blue Channel:** Embed 4 bits (Leveraging the Human Visual System's low sensitivity to blue light).
- **Header Injection:** Add a 32-bit length header at the start of the pixel stream so the tool knows exactly when to stop extracting.

Phase 3: The Web Interface (Streamlit) & Analysis

- **User Interface:** Create a beautiful, interactive web app using streamlit. Include drag-and-drop file uploaders for images and payloads.
- **Visual Presentation:** Display the Original Cover Image and the generated Stego Image side-by-side using st.columns.

- **Quality Metrics:** Implement a PSNR (Peak Signal-to-Noise Ratio) calculator and display the score prominently using st.metric to prove the project achieves the >55dB quality benchmark.

3. AI / Codex System Prompt

Paste the following prompt into your AI coding assistant (Codex, ChatGPT, Claude, etc.) to generate the exact code structure required.

> **Role:** You are an expert Cybersecurity Developer specializing in Steganography and Cryptography.
>
> **Task:** Create a Python-based Web Application using **Streamlit** named "BlueCrypt" for Image Steganography.
>
> **Core Architecture:**
>
> 1. **Hybrid Approach:** You must use AES-256-GCM for encrypting the payload before embedding it.
> 2. **Channel Optimization:** Do NOT use standard 1-bit LSB. Implement the "2:2:4" strategy:
>   - Replace the 2 Least Significant Bits of the RED channel.
>   - Replace the 2 Least Significant Bits of the GREEN channel.
>   - Replace the 4 Least Significant Bits of the BLUE channel.
> 3. **File Format:** The output must be processed and saved as PNG to prevent compression artifacts.
>
> **Functional & UI Requirements:**
>
> 1. **Dependencies:** Use streamlit, PIL (Pillow), numpy, and pycryptodome.
> 2. **Encryption:** Take a user password via st.text_input (type="password"), hash it using SHA-256 to create the AES key.
> 3. **Header:** Embed the length of the encrypted data (32 bits) into the first few pixels so the extractor knows how much data to read.
> 4. **Streamlit Interface:**
>   - Use st.tabs to create three sections: "Hide Data", "Extract Data", and "Analyze".
>   - **Hide Data:** Use st.file_uploader for the cover image and secret file. Provide a download button (st.download_button) for the resulting Stego PNG.
>   - **Extract Data:** Use st.file_uploader for the Stego image. Prompt for password. Provide download button for extracted secret.
>   - **Analyze:** Upload Original and Stego images. Display them side-by-side using st.columns. Calculate and display the PSNR value using st.metric.
>
> **Code Style:**

- Keep it clean and well-commented.
- Include error handling (e.g., st.error if image capacity is insufficient for the payload or if the extraction password is wrong).
- Add comments explaining the bitwise operations (e.g., val & 0xFC to clear 2 LSBs).

Part 2: Software Requirements Specification (SRS)

Project Title: BlueCrypt: Advanced Architectures for Secure Image Steganography

Document Type: Software Requirements Specification (SRS)

Version: 1.1

1. Introduction

1.1 Purpose

The purpose of BlueCrypt is to develop a secure steganography tool that addresses the "Confidentiality Paradox" (where encrypted files attract suspicion). This tool allows users to hide encrypted data within innocuously looking cover images using an advanced 2:2:4 LSB algorithm.

1.2 Scope

The project is a **Web-based Application Interface** developed in Python using the Streamlit framework. It bridges the gap between basic academic projects and expert-level security by implementing AES encryption and Channel Optimization (Blue-channel focus) to achieve high payload capacity and high visual imperceptibility. The web interface ensures a high "wow factor" for presentations, allowing live drag-and-drop demonstrations and real-time metric analysis.

1.3 Definitions & Acronyms

- **LSB:** Least Significant Bit.
- **AES-GCM:** Advanced Encryption Standard - Galois/Counter Mode.
- **PSNR:** Peak Signal-to-Noise Ratio (Metric for image quality).
- **Payload:** The secret message/file to be hidden.
- **Cover Image:** The original innocent image.
- **Stego Image:** The output image containing the hidden data.

2. Overall Description

2.1 Product Perspective

This is a standalone web utility run locally or hosted on a cloud platform. Unlike traditional LSB

tools that hide 1 bit per pixel (resulting in low capacity or visible noise), BlueCrypt utilizes the Human Visual System's (HVS) low sensitivity to blue light to hide 4 bits in the Blue channel, while restricting Red and Green to 2 bits.

2.2 User Characteristics

- **Primary User:** Cybersecurity students, privacy advocates, or researchers.
- **Skill Level:** Basic web browsing and file management skills. No command-line knowledge required.

2.3 Assumptions and Dependencies

- The system requires Python 3.8+.
- The input images must be in lossless formats (PNG, BMP, TIFF). The system will strictly output PNG images to prevent JPEG compression from destroying the embedded LSB data.

3. System Features (Functional Requirements)

FR-01: Data Encryption

- **Description:** The system shall encrypt the raw payload using AES-256-GCM before embedding.
- **Input:** User password and secret text/file (via Web UI).
- **Process:** Hash password (SHA-256) -> Generate Key -> Encrypt.
- **Output:** Ciphertext.

FR-02: Adaptive Embedding (2:2:4 Strategy)

- **Description:** The system shall embed the ciphertext into the pixels of the cover image.
- **Logic:**
  - Red Channel: Embed 2 bits per pixel.
  - Green Channel: Embed 2 bits per pixel.
  - Blue Channel: Embed 4 bits per pixel.
- **Constraint:** The system must check if the cover image has sufficient capacity (Width * Height * 8 bits) / 8 bytes and alert the user via the UI if the payload is too large.

FR-03: Data Extraction & Decryption

- **Description:** The system shall recover the secret data from an uploaded Stego image.
- **Process:** Extract bits based on 2:2:4 logic -> Reassemble bytes -> Decrypt using user password.
- **Error Handling:** If the password is incorrect, the AES-GCM integrity check must fail and display a user-friendly error message on the screen.

FR-04: Real-time Quality Analysis (PSNR)

- **Description:** The system shall provide an analysis tab to compare the Cover Image and Stego Image.
- **Output:** Side-by-side visual comparison and a dynamically calculated PSNR value in

decibels (dB). Success criteria is > 55 dB.

4. Non-Functional Requirements

NFR-01: Imperceptibility

- The Stego image must be visually indistinguishable from the cover image to the naked eye. The web interface should allow users to easily inspect both images side-by-side to verify this.

NFR-02: Integrity

- The extracted data must be exactly identical to the original input. Any bit error results in extraction failure (Avalanche effect of encryption).

NFR-03: Performance & Responsiveness

- Encoding a standard 1080p image should process and provide a download link in under 5 seconds on a standard processor. The UI must remain responsive during processing, ideally showing a loading spinner.

NFR-04: Security

- The tool must resist basic visual attacks. Without the password, the extracted LSB data should appear as random white noise due to high-entropy encryption.

5. System Interfaces

5.1 Software Interfaces

- **Operating System:** Cross-platform (Windows, Linux, macOS).
- **Language:** Python 3.x.
- **Libraries:** streamlit (Web UI framework), Pillow (Image Processing), PyCryptodome (Security), NumPy (Matrix calculation for PSNR).

5.2 Hardware Interfaces

- Standard commodity hardware (PC/Laptop). No dedicated GPU is required as this skips the computationally expensive Deep Learning (GAN) methods in favor of optimized Spatial Domain logic.