Image-Based Text Steganography Using Basic Python

Mohamed Hesham Ahmed
ID : 202401263

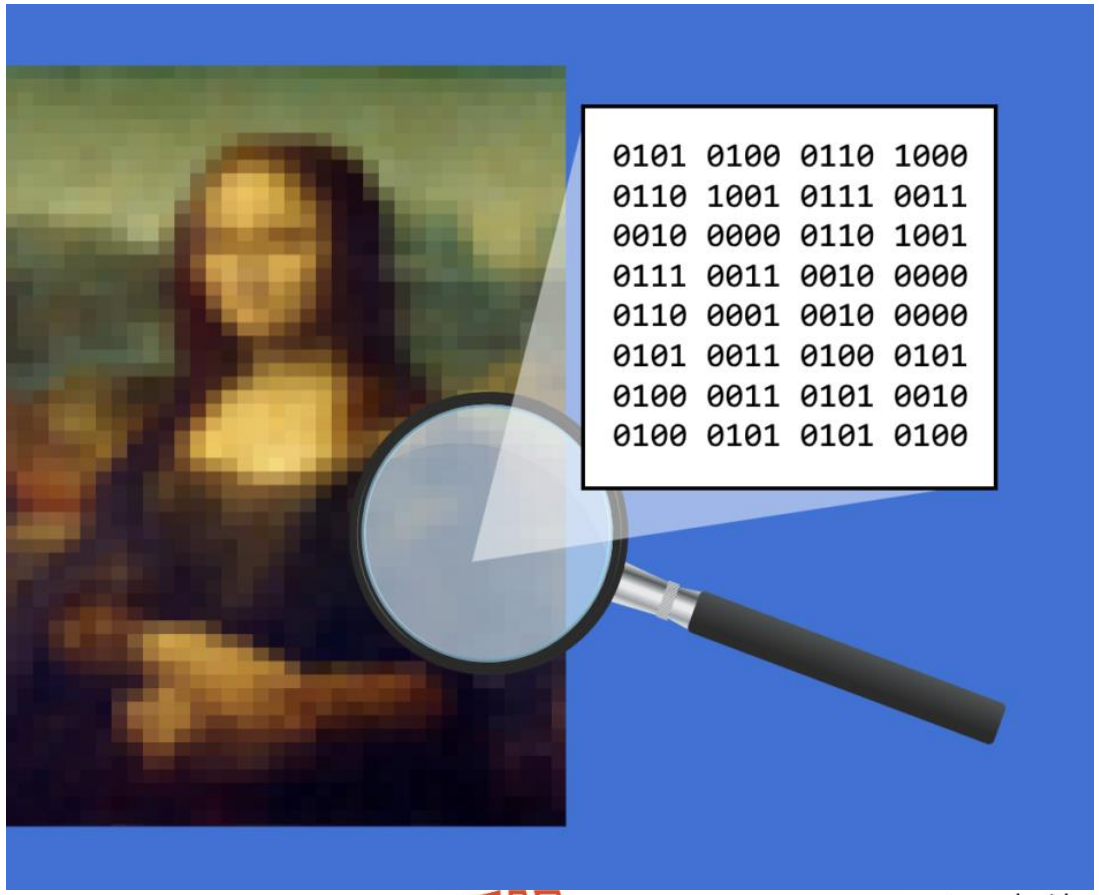
Coursework Title: Steganography Application – Python Implementation

School of Computing

The Knowledge Hub Universities
partnered with Coventry University

What is steganography?

- Came from Greek work ‘steganos’ and ‘graphein’
- It is not cryptography



Project purpose:

Core Objective: Implement image-based text steganography using pure Python without external libraries

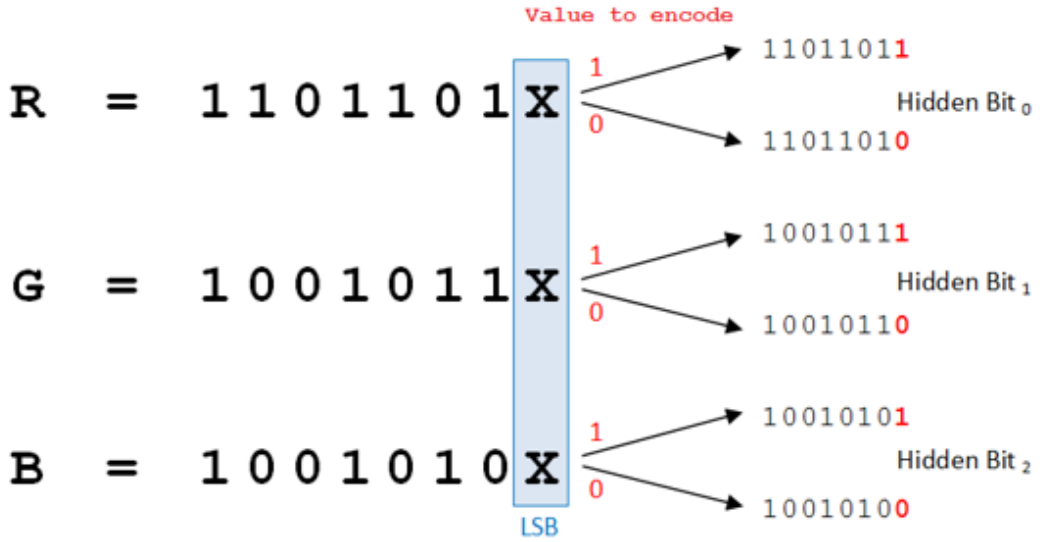
Primary Technique: Hide secret messages in BMP images using LSB (Least Significant Bit) modification

Key Constraint: No external or built-in libraries - rely solely on fundamental Python operations

Technical Focus

How LSB Steganography Works

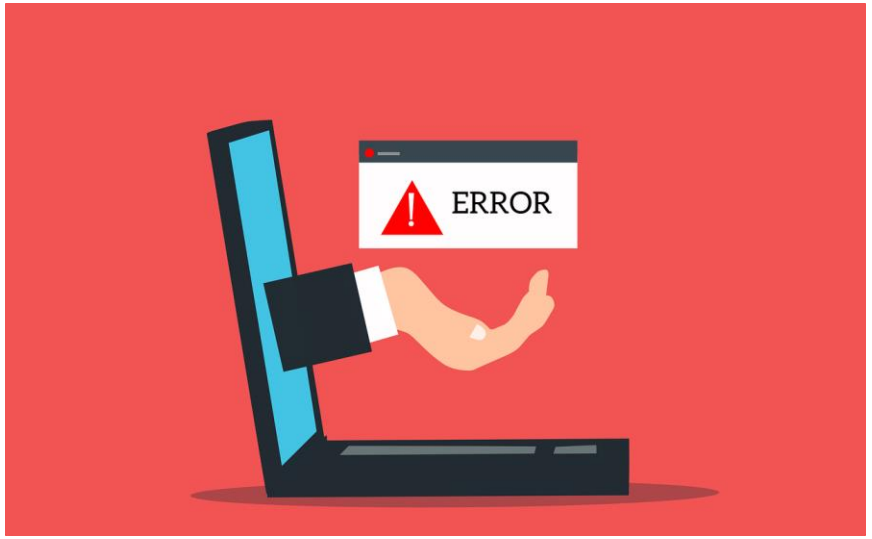
- **Encoding**: Converts the secret message to binary, then replaces the LSB of each pixel's color channel with message bits.
- **Decoding**: Extracts LSBs from the image, groups them into bytes, and reconstructs the original message.
- **Process**: Starts from byte 54 (after BMP header) to avoid corrupting the image.
- **Delimiter**: Uses 1111111100000000 to mark the end of the message during extraction.



- BMP Advantage:
  - Lossless format
  - Simple structure
  - 24-bit = 3 bytes per pixel (B, G, R)
- Capacity: Each pixel can store 3 bits (one per channel)

Error Handling Examples

- If the user entered a wrong password → the program will tell him wrong password, try again.
- If the user uploaded text file that doesn't exist,→ the program will tell him file {filename} not found.
- If the user made a choice that doesn't exist, → the program will tell him Invalid choice please enter 1 or 2 or 3
- If the user entered an empty text file or an empty message → the program will tell him this is an empty message write anything to encode.
- If the user made the output image name, the same as the input image name → the program will tell him input image cannot be the same as output image.
- If the user entered an undefined character not in ASCII table, → the program will tell him the message contains undefined characters.
- If the user entered the path or the name for the images that doesn't exist → File {image name} not found.
- If the user uploaded another image than BMP → the program will tell him that this is not a bmp image.
- If the user entered a message that is bigger than image size → the program will tell him the message is too long that cannot fit the image.



Unit Testing Table

Test ID	Test Description	Input	Expected Output	Actual Output	Status
UT1	Encode short message	“Hello”	messages were hidden successfully	Works	pass
UT2	Decode message	Encoded image	“Hello”	Works	pass
UT3	Empty message	“”	Empty messages write anything to encode	Works	Pass
UT4	Message exceeds capacity	20000 chars	message is too long that cannot fit the image	Works	Pass
UT5	Invalid image type (extension)	wrong.png	This is not a bmp image.	Works	Pass
UT6	Special characters	“Hello@123”	messages were hidden successfully	Works	Pass
UT7	File doesn’t exist	Bridge.txt	Bridge.txt doesn’t exist	Works	Pass
UT8	Message contains characters not in ascii table	اهلا	message contains undefined characters	Works	pass
UT9	Wrong password	Dthew245	Wrong password	Works	pass
UT10	File input	File.txt	messages were hidden successfully	Works	pass

Version Control Repository:

GitHub URL: https://github.com/booka214235/Steganography.git

ppt URL: https://github.com/booka214235/Steganography.git

Interesting or Unusual Choices

- **using custom delimiter:** using 1111111100000000 for smart looping instead of 00000000 to avoid misunderstanding.
- **Manual byte editing:** each byte is extracted and edited without any type of libraries.
- **Password protection:** for extra security.
- **Dual input method:** this gives the option for the user to upload file or enter the message.
- **BMP design:** BMP was chosen because any other image extension would use an external library. Also, we started editing from byte 54 since the first 54 bytes are used as a BMP header, and if we mess with them, the image won't open.

Problems Solved

1- **Capacity Management:** The program compares the number of bits of the secret message and compares it with the number of bytes of input image.
2- **File Corruption Prevention:** The program validates image structure by checking if it is a BMP image by checking if the first two bytes are 'BM' and by skipping the first 54 bytes of the image (header of BMP).
3- **Input Validation:** The program checks for ASCII range (0,127) and file existence.
4- **Empty message:** checking for empty messages by comparing the input with empty string
5- **Delimiter:** Using 1111111100000000 as delimiter to stop the loop
6- **Error messages:** Act as indicators to direct users
7- **User interface:** Simple interface gives full option and control

Time Complexity

Encoding: **O(n)** — where *n* is the message length in bits.

Decoding: **O(m)** — where *m* is the number of bytes processed until delimiter.

Both are **linear and efficient** for practical use.

Thank you