Steganography Project – Group 139

CS 361: Introduction to Computer Security

Ricardo Sanchez
Melissa Mendez

How to Run

In terminal:

> javac Steganography.java
> java Steganography

Introduction

This project implements the steganography technique to hide messages within images. Obviously, it is best to use an original non-public image for best secrecy but for the sake of this program, we have provided simple small colored images from the internet if the user is inclined to use them. We decided to code our technique in Java which includes handy libraries to read/write/save images directly. We used specifically PNG format because it is the only one that uses 4 components: Alpha, Red, Green, and Blue.

Brief Explanation of LSB

The steganography algorithm we used for this project was the **L**east **S**ignificant **B**it technique. A simple and easy-to-code technique. With PNG file format, we took advantage of its ARGB components: *alpha (transparency)*, *red*, *green*, and *blue*. With four components to manipulate, the least significant bit of each can be changed without noticeably altering the original image. That way, we use 4-bits per pixel and use two pixels (8-bits) to hide one character of the message. Hence, in our algorithm a character of the hidden message is represented by 8-bits (two adjacent pixels).

Example

The program gives the user the option to either read and/or write a message from/to an image. After choosing an option, the user will need to provide a *path* to the PNG file they wish to use. If the user doesn't have a PNG file, (s)he can leave the *path* blank and the program will use the default image (located inside the "images" subfolder). In the end the program will either provide the hidden message, in the case that the user chose to read a message from an image; or it will tell the user that the message was hidden successfully, in the case that the user chose to hide a message. The image with the hidden message will be saved in the same location as the original image, but with a different name (i.e. "(hidden message)" will be appended to the original name of the PNG file).

1. In the following example the user hides a somewhat long message in “tiny.png” and the program notifies him/her that only a portion of the message was written because the image is too small.
2. Then the user successfully tries to read the message that (s)he just wrote in “tiny(hidden message).png”.
3. Lastly, the user exits the program.

```
jects\Project01\src> javac Steganography.java
PS C:\Users\Ricardo Sanchez\OneDrive\UT\08 - Spring 2018\Intro to Security\Pro
jects\Project01\src> java Steganography
Available options:
  - [H]ide a message in an image.
  - [R]ead a hidden message from an image.
  - [E]xit.
Choose an option: H
Enter path to the image you want to use: images\tiny.png
Image converted to support aRGB (transparency).
Enter the message to hide in the image: Hello there, I hope that you are havin
g a very pleasant day! Good bye!
Unfortunately, the image used doesn't have enough pixels to fit all of your me
ssage, but we were able to hide the following part: "Hello there, I hope that
you are having a very ple".
```

1.

```
Available options:
  - [H]ide a message in an image.
  - [R]ead a hidden message from an image.
  - [E]xit.
Choose an option: R
Enter path to the image you want to use: images\tiny(hidden message).png
Hidden Message: Hello there, I hope that you are having a very ple
```

2.

```
Available options:
  - [H]ide a message in an image.
  - [R]ead a hidden message from an image.
  - [E]xit.
Choose an option: E
PS C:\Users\Ricardo Sanchez\OneDrive\UT\08 - Spring 2018\Intro to Security\Pro
jects\Project01\src>
```

3.

Flow of Action

- Ask user if want to **read** or **hide a message** from/to an image or exit the program.
- **To hide a message** in image:
  - Ask for *path* of the PNG image to use.
    - If the user doesn’t specify a path (leaves it blank), then use the default image.
  - If the provided image does not support the aRGB color model, convert it so that it supports the aRGB color model. We do this so that we can use the Alpha component.
  - Ask for message to hide in the image.
  - Start from upper left most corner of image, iterate from left to right, top to bottom of image and write message.
    - During loop, check if exceeded message length or image height/width
    - Call write_to_pixels function call

        - In loop:
            - Will write for **two** pixels at a time
            - Get color for the pixel currently and the adjacent one using x,y coordinates of image passed through call
            - For each aRGB component:
                - Shift the aRGB value to extract each component individually.
                - Use bitwise AND to extract and clear the LSB
                - Use bitwise OR to write data in the LSB
                - Write the modified aRGB values to the image.
    - After loop, write "message-terminating" character – "11111111".
    - If the message is too long to fit in the image, then the program will stop writing the message whenever it reaches the last pixel of the image (i.e. it will crop the message). You can use the tiny.png image inside the "images" subfolder to test this behavior.
    - Save image as *<name>(hidden message).png*
    - Notify user of success/failure.
- **To read a message** from image:
    - Ask for *path* of the PNG image to use.
        - If the user doesn't specify a path (leaves it blank), then use the default image.
    - Check if image supports aRGB, if not won't read the image since our algorithm uses aRGB.
    - Start from upper left, iterate from left to right, top to bottom of image and read message
        - During loop, check if exceeded image width/height.
        - Call read_from_pixels function call:
            - Get the aRGB value of the two adjacent pixels from the image.
            - Read in Alpha first, then Red, Green, Blue LSB by decrementing the shift each iteration. Append the LSB to a string.
            - In the end the string will have 8 characters, one for each bit.
            - Convert the binary string from binary to decimal and return that.
    - Returning from call:
        - If the return value equals the null-terminating character (i.e. "11111111") then exit loop
        - If the return value is out of ASCII bounds, the data is most likely garbage and holds no human readable message, so set error boolean and exit
        - Else append the character to the string to return the full message.
    - After loop, if it contained a hidden message, print message. If not, notify the user that the provided image contains no hidden message.
