Steganography Over Multiple Cover Images

Mayur Mehta
Mitchell Burgunder

12-11-15

Purpose

The purpose of this project is to learn the benefits and drawbacks of a distributed approach to steganography. The goal of the project is to design and implement a steganography system and test it for robustness, detectability, and practicality. The final part of the project is to compare the implementation with against other research and draw conclusions.

Experiment

After we implemented the design, we tested for robustness, detectability, and practicality. For robustness, we tested the application for against different types of inputs for both cover files and secret files. In order to test the detectability of our implementation, we used a steganalysis tool. Finally, for practicality, we tested the overall responsiveness and runtime of our application.

Results

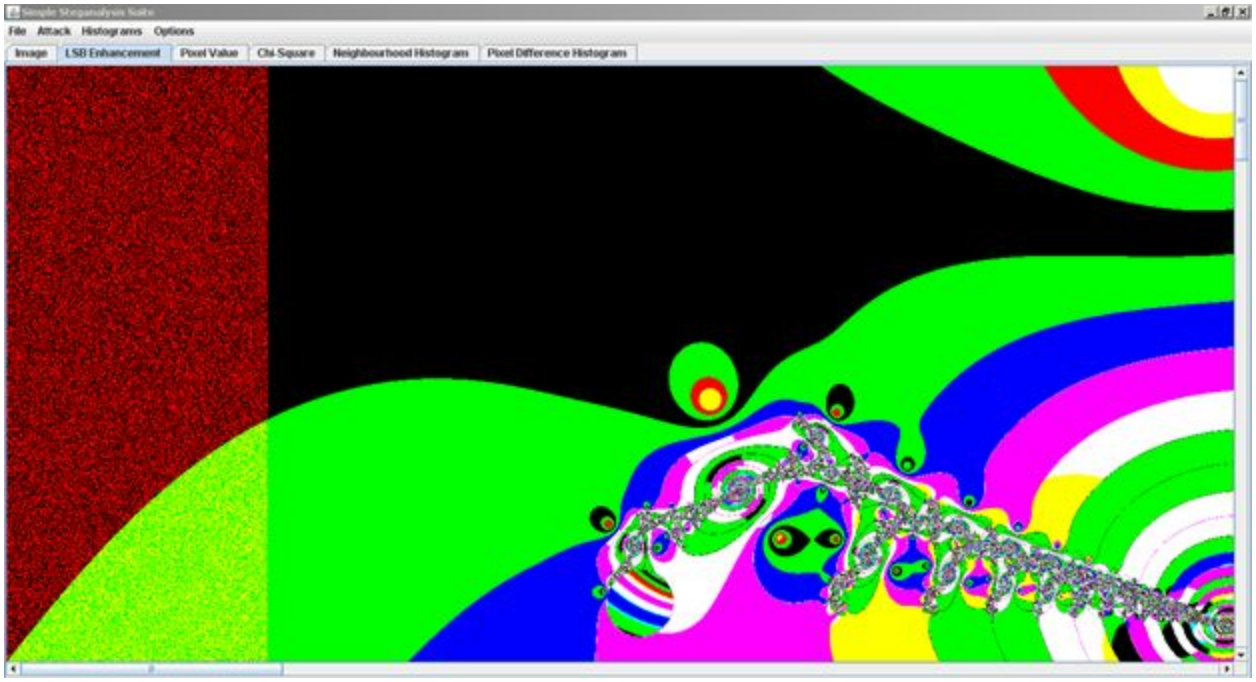

Figure 1: An LSB enhanced image of a stego file using the Simple Steganalysis Suite

Analysis of Results

We tested our system by first embedding an audio (mp3) file in an image (png) file. Then we used the Simple Steganalysis Suite to try to detect our steganography technique. As can be seen in the figure above, the steganalysis software was able to detect our steganography technique by enhancing the least significant bits of the image. In the figure, we can see where the audio file is embedded in the image (the noisy region to the left). Based on this and other research, it seems that LSB steganography can be fairly easy to detect.

However, there are some advantages to a distributed steganography system. Our implementation allows the user to embed larger secrets by partitioning them into smaller pieces. It also adds another layer of difficulty for the attacker because the decoder must have the pieces in the right order in order to recover the secret. Finally, with AES encryption, we can guarantee the confidentiality of the secret.

Security Considerations

In order to provide message confidentiality, the secret message/ file is encrypted before it is encoded into a cover file. We use AES symmetric key cryptography with 128 bit keys for the encryption. Since user passwords can be weak, we use the Random function provided by the PyCrypto library to generate 128 bit keys. This function implement a secure random number generator and is used to generate the IV for AES. The generated key is outputted for the user whenever a secret file or message is encoded with our system. This key should be securely distributed to the recipient in order to properly recover the secret from the stego files.

The steganography system that was implemented used least significant bit (LSB) steganography to embed secrets in cover files. This system provides good anonymity against casual observers, but is not effective against steganalysis techniques. In the case where the secret is detected, it still cannot be deciphered without the correct encryption key and order of images. The distribution of the secret among many cover files provides some benefits. The attacker must have all stego files and the encryption key in order to recover the secret. However, it should be noted that if the stego images are posted in different locations, it becomes easier to identify the creator via an intersection attack. The main benefit of a distributed approach to steganography is that it becomes easier to work with larger secret. With this system, we do not need a large cover file if the secret is large. We can instead distribute partitions of the large secret in multiple smaller cover files.

User Documentation

The distributed steganography system is implemented in python and uses the python image library (PIL) and PyCrypto library. PIL is used for image processing (opening images, reading pixel values, and writing images). PyCrypto is used for its implementation of secure random number generation and AES encryption. The entire implementation is define in the stego.py file.

The program can be run by typing python.exe stego.py. No other arguments are needed because the program uses a menu system to make it more interactive. The program will first ask if the user want to encode or decode. Then it will ask the type of secret to encode or decode. We support either a simple message that the user can input or any type of secret file (pdf, mp3, etc.). Then the user must type the names of the cover files or stego files depending on whether they are encoding or decoding. All files must be in the project directory in order to be detected by the program. When the user is done inputting filenames, the user can enter “done” to move on to the next step. If encoding was selected, the user will enter the secret message or the name of the secret file to be encoded. Then the secret will be encrypted and embedded in the cover images to produce stego output images. The key will be generated using a secure random number generator and outputted for the user. If decode was selected then the user must enter the key and a name for the output file (if applicable).

Method Summary

- encrypt(message)
    - encrypts the given message using AES 128 bit symmetric key encryption
- embed(cover_file, secret)
    - embeds a secret (as a string) into a specified cover file in least significant 2 bits of Red channel
    - Performance: O(m*n) where m and n are the dimensions of the image
    - Space: O(m*n) where m and n are the dimensions of the image
- message_encode(message)
    - converts the message (as a string) into an array of bytes
    - Performance: O(n) where n is the size of the message
    - Space: O(n) where n is the size of the message
- decrypt(cipher_message, key)
    - attempts to decrypt the given message using the specified key
- recover(stego_file)

    - recovers a secret message from the least significant 2 bits of the specified stego file
    - Performance: O(n) where n is the size of the secret
    - Space: O(m*n) where m and n are the dimensions of the image
- message_decode(message)
    - converts an array of bytes back into a string
    - Performance: O(n) where n is the size of the message
    - Space: O(n) where n is the size of the message
- generate_key()
    - creates a cryptographically secure 128 bit key and prints it (in hexadecimal representation) for the user to store for transmission
- split_string(input_string, num_splits)
    - divides the given string into num_splits chunks as equally as possible
    - returns an array of Strings representing the partitions
    - Performance: O(n) where n is the number of splits
    - Space: O(n) where n is the size of the input string
- main()
    - the menu functionality and user feedback

Design

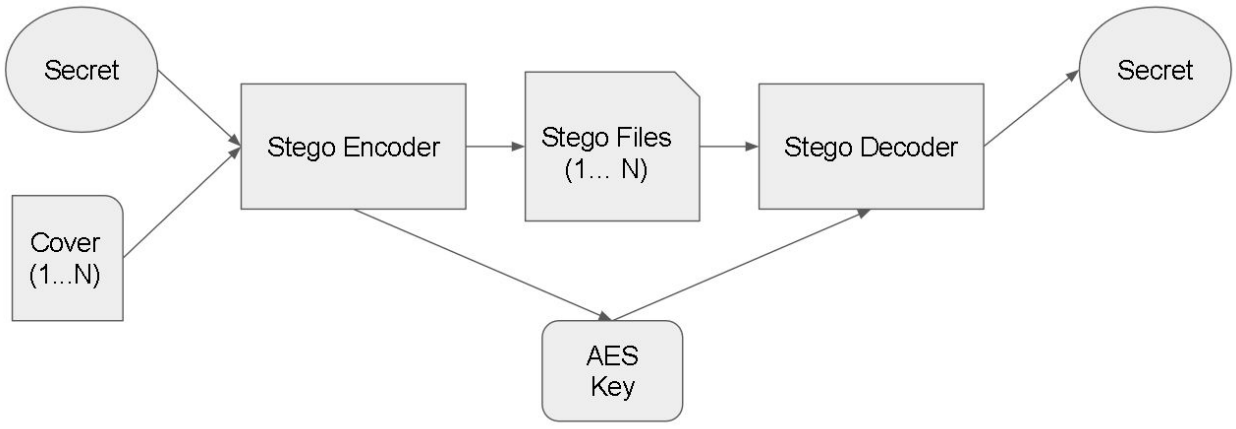

Figure 2: A diagram representing data flow in the system

The Stego Encoder (embed function) is used to encode a bits of a secret into the last two least significant bits of ever Red value in an image. It will take a string as input which represents the secret to be encoded. Then it will call message_encode to convert this string into an array of bytes. The first 32 bits that are encoded represent the size of

the secret so that the user does not need to know the size of the secret in order to recover it with the stego decoder. In order to distribute a large file in multiple cover files, the embed function is called for each partition of the secret.

The split_string function takes a secret represented as a string and returns an array of strings. The secret is partitioned by the split_string function based on how many cover files there are.

The recover function takes a stego image and looks at the first 32 bits to determine the size of the secret that is embedded. It then takes the two least significant bits in the Red values of the image and creates a byte array. Once the whole secret is read, it will call message_decode to convert the byte array to a string and return it.

When the secret is read in as an input in the main function, it is immediately encrypted (AES 128) with a call to the encrypt function. The encrypt function calls the genrate_key function which uses a secure random number generator to create a 128 bit key for the user. This key is printed for the user in hex so that the data can be decrypted in the future. If the user selected to recover a secret from a stego file, the recover function will be called to extract the encrypted secret from the file. The decrypt method will take a string representing the ciphertext and a string representing the key in hex. It will attempt to decrypt the ciphertext and return the plain text to the user.

Testing

The distributed function of the program was tested using one, or more than one cover file. If given a secret message too large for the cover file the program will notify the user that the file was too large. The program was tested using both strings and files (mp3, png, etc.) as input and successfully decoded them from the stego files. If the user does not give the correct key then the message is not returned correctly. If the user does not input the stego files in the correct order then the message is not returned correctly.

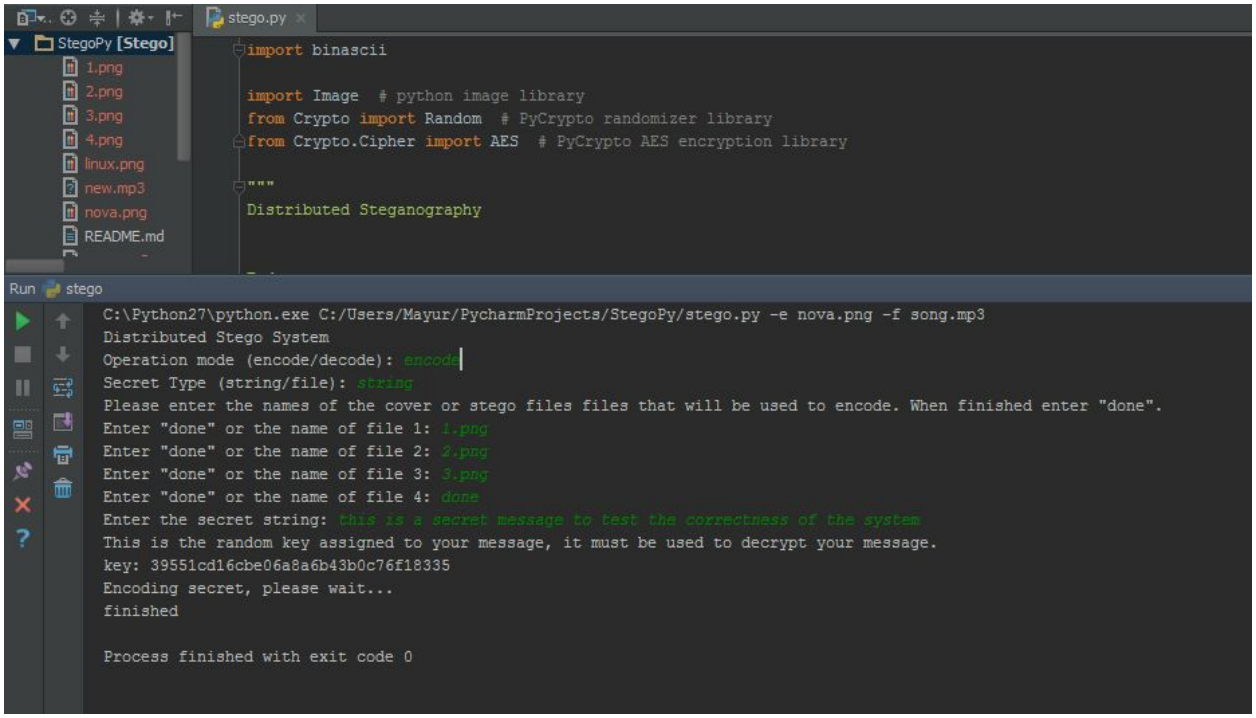

Figure 3: Encoding a secret in three cover files

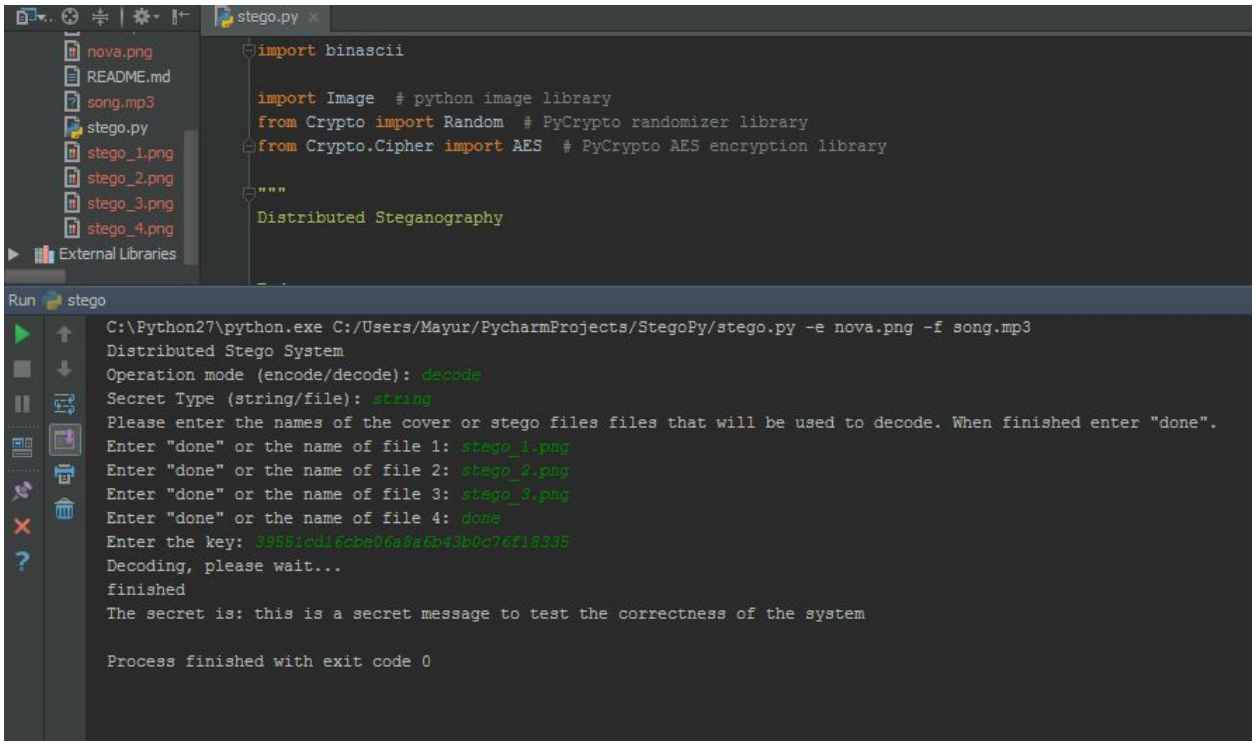

Figure 4: Successfully decoding a secret from multiple stego files

References

- Python Imaging Library, PythonWare, http://www.pythonware.com/products/pil/
- PyCrypto, Dwayne Litzenberger, https://www.dlitz.net/software/pycrypto/
- Simple Steganalysis Suite, beob…@gmail.com, https://code.google.com/p/simple-steganalysis-suite/
- Dickman, Shawn D. "An Overview of Steganography."Department of Computer Science, James Madison University Infosec Techreport (2007).
- Fridrich, Jessica, Miroslav Goljan, and David Soukal. "Searching for the stego-key."Electronic Imaging 2004. International Society for Optics and Photonics, 2004.
- Morkel, Tayana, Jan HP Eloff, Martin S. Olivier. "An overview of image steganography."ISSA. 2005.
- Provos, Niels, and Peter Honeyman. "Hide and seek: An introduction to steganography."Security & Privacy, IEEE1.3 (2003): 32-44.