A Picture's Worth a Thousand Bytes

A Comparison of Spatial Image Steganography Algorithms

Aron Smith-Donovan, Maddie AlQatami, Randy Jose Beidelschies, Colin Kirby

INTRODUCTION

Steganography is the practice of concealing information within other innocuous data, such as images, video, audio, or simple text files. Steganography differs from cryptography in the manner of data concealment: while cryptography conceals information by making it unintelligible, steganography aims to conceal the presence of that information entirely [7]. In comparison to another neighboring field, watermarking, steganography has the capability to transmit large amounts of information, but is not necessarily robust against interference, intentional or otherwise [11].

Image steganography conceals data, known as the *payload,* within an image, known as the *cover* [7]. By manipulating the byte-level data of image pixels, the payload can be embedded into the cover image without resulting in changes detectable to the human eye.

Figure 1: Spot the difference. The upper image is unaltered. The lower image is a cover containing the concealed payload which reads "This is a secret message" when decoded with our naive algorithm, which uses a Least Significant Bit Replacement algorithm.

Image steganalysis is the study of detecting a steganographic image and obtaining the encoded payload. There are many steganalysis techniques in use that utilize statistical and algorithmic methods to determine whether an image has been altered [11].

We implemented two steganographic algorithms for comparison with two different ends in mind: spatial and runtime efficiency vs. security against an adversary equipped with the tools necessary for basic steganalysis. Then, we further improved overall payload security for both algorithms by encrypting the payload before embedding it. In this paper, we will discuss the details of our implementation, user interface, comparative spatial and runtime complexities, and the degree of security that each algorithm achieves.[1]

LITERATURE REVIEW

All image steganography techniques can be defined by their answers to three main questions (although not all of their answers fare well in the face of advancing steganalysis methods). These questions address 1) where the payload is hidden within the image, 2) whether the embedding technique is safeguarded against detection, and 3) how secure the payload is in the case of discovery by an adversary [7].

There are two main techniques used regarding the second question of embedding security: spatial domain techniques, which directly manipulate pixel values, and transform domain techniques, which transform image pixel value frequencies [3]. Spatial domain techniques include *least significant bit* (LSB) [2] and pixel value differencing [10], and transform domain techniques include discrete wavelet transform, integer wavelet transform, and discrete

[1] GitHub repository for our project is available for public access at github.com/colinkirby/SteganographyCapstone

cosine transform [4]. Our discussion and implementations will be limited to spatial domain techniques, as they are characterized by shortened implementation time, non-specialized hardware requirements, and reduced time complexity [13].

We can further categorize the spatial techniques by where they embed the payload data. While blind hide methods embed data sequentially in an image pixel array [1], scatter methods attempt to pseudo-randomly scatter data amongst the pixels of an image by using sequence generators to identify pixels for alteration [7]. Chaos-based adaptive methods identify *regions of interest* (ROI) that, if altered, would result in the least amount of noise distortion [14]. This statistical noise distortion can be identified by steganalysis algorithms that calculate the frequency of and correlation between the color values of pixels that are close to each other [11]. Thus, methods that both utilize ROI and pseudo-randomly scatter the positions of altered pixels perform among the best for steganographic algorithms in the spatial domain against steganalysis [15].

Finally, there are several approaches to ensuring the security of the payload. One approach encrypts the data before embedding, which leads to an increase in the size of the payload value, often by a significant scale factor. Alternatively, the data can be subjected to a chaotic mapping technique which maintains the size of the payload but doesn't ensure that the data is impervious to cryptanalysis if the adversary has sufficient resources [7].

METHODS

Naive Approach with LSBR and Blind Hide

One algorithm for encoding the payload into the cover is *Least Significant Bit Replacement* (LSBR). Each pixel in an image holds an RGB value that is represented by 3 bytes. The least significant bit of the R byte, G byte, and B byte can be flipped (from 1 to 0, or 0 to 1) to match the corresponding bit in the payload, with a resulting difference in the overall color of the pixel indiscernible to the human eye. In the case where the payload is a text file, each ASCII character, represented by one byte, can be hidden across three different pixels. Three pixels are represented by nine bytes in the image file, so we can hide each bit of our ASCII character in the least significant bit of eight of the nine pixel bytes [12].

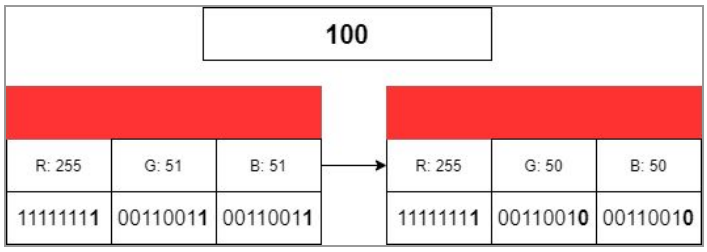

Figure 2: LSBR is used to encode the three-bit binary string, 100, into a single pixel. The LSBs of the RGB bytes in the original pixel on the left are 1, 1, and 1 respectively. In the new pixel, these bits are changed to 1, 0, 0. The LSB of the R byte remains unchanged, as it matches the first bit of the binary string, but the LSBs of the G and B bytes are flipped.

This very subtle change in color is undetectable to the human eye [11], and contains our encoded payload. Hiding data within pixels is only one element of steganography, the method in which pixels are selected for hiding data is arguably more important.

The first method we used to distribute our data across an image, blind hide, sequentially stores data in pixels. Starting at the top-left pixel, bytes of data are stored in blocks of 3 pixels from left to right. Once one row of pixels is filled, the algorithm proceeds to the following row and continues until all data has been stored or until no unaltered pixels remain. The simplicity of this algorithm allows for easy encoding and decoding, assuming both parties know that this algorithm was used. However, its simplicity is its drawback, as it is very susceptible to steganalysis, and can easily be decoded because the data is not encrypted.

Naive Approach (with encryption)

The security of the naive approach can be improved by adding a layer of simple symmetric encryption before the image encoding algorithm. For this method, the data is encrypted with a generated base-64-encoded 32-byte key, returning a secure base-64 encoded token as the output; the key is required to retrieve the original data from the token. The encrypted token is then distributed across the

image using the same sequential naive approach as before. This modification does not reduce the detectability of the data, however if the encoded information were to be detected and derived from the encoded image, the adversary would only have access to the encrypted token, not the original payload data. This adds the additional time expense required to derive the key through brute-force methods in order to retrieve the original data, which additionally discourages unauthorized retrievals.

Fernet Encryption

For this study, we used Fernet encryption from the Python cryptography library for symmetric encryption. Fernet uses the *Advanced Encryption Standard* (AES) in *Cipher Block Chaining* (CBC) mode with a 128-bit key for encryption using current *Public Key Cryptography Standards* (PKCS7) padding. A randomly allocated salt value is also used to make encryption more secure. Additionally, Fernet utilizes a secure method for generating keys, timestamps encrypted messages, and uses *Hash Message Authentication Code* (HMAC) and the 32-bit word size *Secure Hash Algorithm 256* (SHA256) for signing messages and detecting tampering. A Fernet key (visualized in *Fig. 3*) contains a signing key used to sign the HMAC and a private key used by the encryption protocol.

Signing-key 16 bytes	Encryption-key 16 bytes

Figure 3: Structure of a Fernet key [6]

The encrypted message is represented by a Fernet token (as shown in *Fig. 4*). It contains an 8-byte timestamp, a 16-byte initialization vector used for AES encryption and decryption, and a 32-byte signed HMAC using the signing key of the Fernet key. The ciphertext contains the encrypted version of a plaintext message and is padded to always be a multiple of 16 bytes in length [6].

Version 1 byte	Timestamp 8 bytes	IV 16 bytes	Ciphertext n*16 bytes	HMAC 32 bytes

Figure 4: Structure of a Fernet token [6]

Chaos-based Edge-adaptive Approach using Canny Edge Detection and LSBM (with Encryption)

The chaos-based edge-adaptive approach developed by Roy et al. [7] and adapted for text payloads improves upon the naive approach in several ways. The probability of successful steganalysis is reduced by only performing LSB alterations on a set of pixels that are pseudo-randomly scattered throughout the image. Furthermore, certain areas of a cover image, ROI, are less susceptible to common methods of steganalysis [7]. One of these ROIs is the edge pixels in an image.

These pixels are selected by applying *Canny's Edge Detection* algorithm to the original image to identify edge pixels, and only encoding payload data into pixel values within the set of identified edges. Canny's method has a high threshold for noise and can detect true weak edges [16], making it ideal for isolating regions of low potential noise distortion [7]. Furthermore, it is considered the standard optimal method for detecting edges in images [17, 18].

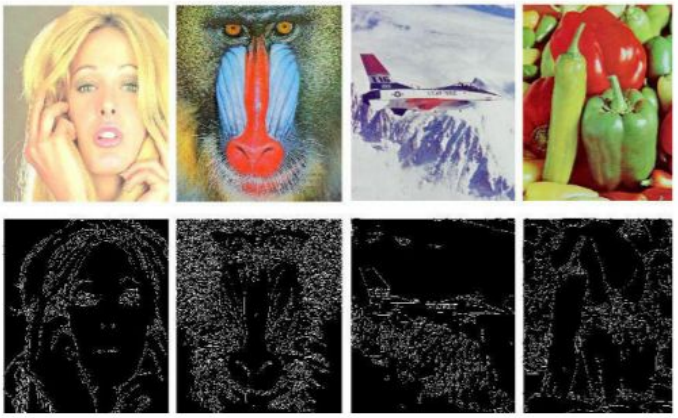

Figure 5: Cover files and their corresponding edge pixels identified by Canny's Edge Detection algorithm using threshold values chosen to maximize edge significance [7]

Additionally, this combined algorithm reduces the number of pixel values that require alteration overall by performing the embedding with a combination of *matrix encoding* [9] and *LSB Matching* (LSBM) [8, 7]. The combined algorithm modifies pixel LSBs according to the following condition:

For bits a_1, a_2, and a_3, corresponding to the LSBs of the R, G, and B components of a modifiable pixel respectively, and bits x_1 and x_2 of the payload to be encoded, we either change one of a_1, a_2, or a_3 or leave all three unaltered to fulfill two equalities, where $\oplus$ represents the binary operator, "exclusive or".

$$x_1 = a_1 \oplus a_3$$
$$x_2 = a_2 \oplus a_3$$

Figure 6: Necessary conditions for encoding using a combined matrix encoding and LSBM algorithm [7]

This can be accomplished by modifying the bit values as follows.

Condition	Action to be taken
$x_1 = a_1 \oplus a_3$ $x_2 = a_2 \oplus a_3$	No change required
$x_1 = a_1 \oplus a_3$ $x_2 \neq a_2 \oplus a_3$	Change component G to match conditions (3) & (4)
$x_1 \neq a_1 \oplus a_3$ $x_2 = a_2 \oplus a_3$	Change component R to match conditions (3) & (4)
$x_1 \neq a_1 \oplus a_3$ $x_2 \neq a_2 \oplus a_3$	Change component B to match conditions (3) & (4)

Figure 6: Bit modification algorithm for encoding [7]

This methodology allows for two bits of payload data to be encoded within a single pixel while retaining the integrity of at least two components of the pixel color value. Finally, as with the improved naive approach, both the original message and the set of edge pixels are passed into the encryption algorithm and outputted as an encryption key.

Graphical User Interface

Our GUI implementation has two options for encoding and decoding images, our LSBR algorithm with encryption and our chaos-based algorithm with encryption. For encoding, the GUI allows the user to upload both an image, to serve as the cover, and a text string, to serve as the payload. The text can either be manually entered through keyboard input (see *Fig. 7)*, or a file can be uploaded using a *browse files* prompt, restricted to .txt files (see *Fig. 8*). For image upload, the GUI prompts users to browse their files and accepts only files with the .PNG file extension (see *Fig. 9)*.[2]

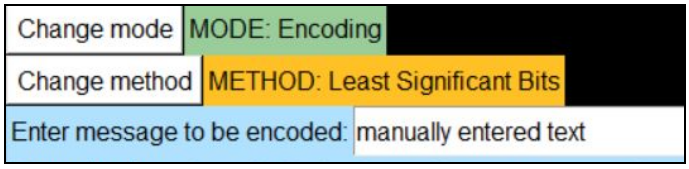

Figure 7: Manually entered payload text

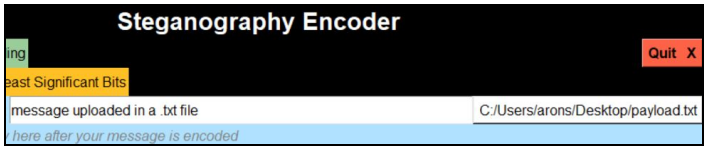

Figure 8: Payload text uploaded from a .txt file

Figure 9: The GUI after uploading an image file

After encoding, the GUI allows the user to download two new files to the parent directory of the provided image file (see *Fig. 10*): a cover image containing the payload with an updated file name and a text file containing the associated Fernet encryption key (see *Figs. 10, 11*). These two components can be reuploaded to the GUI after switching to decoding mode in order to obtain the original message (see *Figs. 12, 13*).

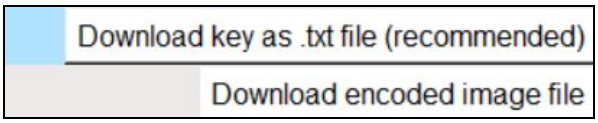

Figure 10: The download prompts for the user

[2] We use only the .PNG extension because the .JPG and .JPEG extensions utilize a lossy compression algorithm that alters byte-level values of pixels.

Figure 11: The encoded image and the associated key downloaded as local files to the user's CPU

Figure 12: The GUI after uploading the encoded image and associated key and retrieving the original data

Figure 13: The decoded data downloaded as a local .txt file to the user's CPU

RESULTS & DISCUSSION

Our algorithms were successful in encoding a payload into an image, and later retrieving it. However, we remain limited in payload size by the dimensions of the cover image. Additionally, our different algorithms performed with varied runtime performance reflective of their relative complexity. Most notably, our three algorithms differed drastically in regards to the relative security against steganalysis of the payload encoded, with only our chaos-based approach secure against both discovery by steganalysis and decryption, if discovery of transmission were to occur.

Spatial Limitations

Assuming that the user uploads a cover image with dimensions 400×400, we can expect to see different maximum payload sizes for each of our steganographic algorithms.

With our naive approach, we can encode a byte of payload data into 3 pixels. This translates to approximately 53 KB of data for an image with dimensions 400×400:

$$\frac{400 \times 400}{3} = \frac{160{,}000}{3} = 53{,}333 \text{ bytes.}$$

For our naive approach using encryption, the calculations are not as simple because the token system used by Fernet is of variable length. For a message of length n bytes, the resulting Fernet token will be of length $16n + 57$ bytes (see *Fig. 4*). From the calculations for the naive approach, we know there are sufficient pixels to encode 53,333 bytes total. Thus we can expect to encode a significantly reduced maximum payload of 3.3 KB of unencrypted data:

$$\frac{53{,}333 - 57}{16} = 3{,}329 \text{ bytes.}$$

For our chaos-based algorithm with LSBM, each pixel can encode two bits, or a quarter of a byte, of payload data. However, because our chaos-based approach scatters altered pixels throughout the cover image, the encoding capacity of our cover image is dependent on the percentage of pixels identified by the Canny edge detection algorithm as edge pixels. For a 400×400 image, we expect to have a capacity for

$$\frac{160{,}000}{4} \times \alpha \times (1/\beta) = 4{,}000 \times \alpha \times (1/\beta) \text{ bytes}$$

of data, where α represents the proportion of pixels identified as edges by the Canny edge detection algorithm and β represents the bit scale factor between the length of the original payload and the Fernet encrypted payload. We can further simplify this calculation with the knowledge that the recommended threshold values to identify only significant edges, passed as parameters to the Canny edge detection algorithm, result in approximately 5% of pixels identified as edge pixels [5]. Finally, using the same calculations to determine the maximum

message input size to the Fernet algorithm we can see that only

$$\frac{4{,}000 \times 0.05 - 57}{16} = \sim 121 \text{ bytes}$$

of the original payload can be encrypted then encoded using our chaos-based algorithm.

This is notably insignificant, as 121 bytes is less than half of the maximum length for a Tweet. In order to encode 1 KB of payload data (before encryption) using our chaos-based edge-adaptive algorithm, we would need to use an image of dimensions at least 1200×1200.

Time Complexity Analysis

A comparative complexity analysis of our three approaches is surprisingly straightforward. As to be expected, our naive approach using LSBR is completed in $O(n)$ time, where n is the size of the payload in bytes. Furthermore, standard symmetric encryption algorithms, including Fernet, perform in $O(n)$ time as well [6]. Unfortunately, Canny edge detection is somewhat less efficient. For image dimensions $w \times h$, edge pixels can be identified in $O(wh \times \log(wh))$ time [5]. However, neither matrix encoding nor LSBM increase the runtime for payload embedding beyond $O(n)$ [7]. Therefore, our chaos-based approach runs in $O(wh \times \log(wh))$ time, which is still perfectly acceptable for our application, where users encode a single cover image at a time, presumably for personal use.

Relative Security

Our naive approach using LSBR and Blind Hide is both insecure and easily detectable by *steganalysis* [12]. Standard steganalysis techniques calculate the correlation between nearby pixel color values of unaltered images to identify the *noise floor*, or expected level of white noise in pixel color values of similar images. Then, the noise profiles are calculated for images identified for steganalysis and these profiles are compared with the expected noise floor [11]. Images altered using blind hide are easily identified as steganographic images from their abnormal noise profiles, which can be identified due to the "*Step Effect"* these abnormal noise profiles have on pixel difference histogram analysis [19]. Then, furthermore, the simple sequential LSBR embedding scheme renders payload extraction trivial after a steganographic image has been flagged.

By additionally implementing encryption on top of our LSBR and blind hide algorithm, we ensure that our payload remains secure despite the high risk of detection when faced with potential adversaries. However, steganography is distinct from cryptography in that it aims to conceal the presence of payload transmission entirely. Thus, the users of our application likely want to avoid detection entirely, as the discovery of covert communication, even if the contents of the communication cannot be revealed, can be damaging to the communicators in and of itself. Therefore, the payload should be both secure and undetectable within the steganographic image.

Our chaos-based edge-adaptive method is both secure and difficult to detect by steganalysis [7]. Steganography embeddings using LSB alterations are vulnerable to several types of steganalysis: χ^2-testing using *Pair of Value* (PoV) steganalysis [8], and detection of the Step Effect [10]. PoV steganalysis identifies all pairs of subsequent LSB pixel values and determines using χ^2- testing whether the distribution of pixels with those two LSB values are uneven (the expected result for an ordinary image) or statistically close to a 50-50 distribution (the result if LSBR steganography is implemented on the image). By utilizing LSBM, our chaos-based approach demonstrates a significantly low probability of discovery when subjected to χ^2-steganalysis ($P = 6.0976 \times 10^{-4}$) [7]. This demonstrates that the apparent random distribution of the image pixels is not disturbed in a statistically significant way due to our embedding method. Furthermore, the Step Effect is alleviated by our edge-adaptive approach, which does not alter smooth-pixel-value regions of a cover image.

CONCLUSION & FUTURE WORK

The algorithms implemented have shown, through our analysis, to be successful in encoding and retrieving payloads, but differ in the extent of their limitations. The shortcomings of our final product lie in the limited payload size, imperfect robustness to steganalysis [20], and suboptimal expected runtimes. As a result, we recommend that our implementation not be used for any purpose that requires complete confidentiality or involves especially sensitive information. If we were to continue the project from this point, given further time and resources to do so, our future work would focus on mitigating the impact of these limitations before moving toward expanding the base functionality.

Improving payload capacity

We found that both our naive and chaos-based approaches are significantly limited as to the maximum size of the payload that can be encoded, for chaos-based with encryption only about 121 bytes for an image of dimensions 400×400. We first acknowledge that storage limitations are inherent to the study of image steganography: because the integrity of the original depiction must be maintained, each pixel is limited to the extent of its data that we can repurpose for our encoding, and there exists in any image a finite number of pixels. As such, we would be unable in any circumstance to fully alleviate considerations of payload capacity, and so focus rather on potential modifications for relative expansion, bound by image size.

One factor which is, at present, a major contributor to payload capacity limitations is the use of encryption before encoding a given source text. Because of the simplicity of our naive approach and its susceptibility to steganalysis, encryption is a rational addition to further protect encoded user data; however, by nature the encryption increases the length of the payload to be encoded. Although removing encryption would increase the vulnerability of messages once interference has been detected in the image, we propose a two-pronged approach to increase payload capacity: the first modification is to improve the chaos-based approach with respect to steganalysis, and the second is to remove the encryption protocol. By improving the encoding itself to be more robust to steganalysis attacks, we then eliminate the need for our encryption protocol, as we can reasonably expect the encoded payload to not be detected by unauthorized parties. In addition, the encoding scheme can be modified to increase payload capacity.

Improving the encoding approach

Our chaos-based edge-adaptive approach was shown to require four pixels to encode a byte of payload data, slightly less space-efficient than the three pixels per byte required for our naive approach, and is further limited by the availability of edge sets. In the probable circumstance of an image with few derived edges, the payload size is further limited. Consequently, we posit two potential techniques to increase our algorithm's maximum payload size for a given image: we could 1) utilize a chaotic scattering technique to embed the data and remove the encryption component from our algorithm entirely, or 2) implement a chaotic mapping technique to secure our payload in the case of detection in place of the symmetric encryption protocol.

A chaotic mapping approach for securing the payload in case of discovery would transform payload data using a Cat map, as was successfully implemented for further security by Roy et al. and Mishra et al. [7, 21]. Alternatively, a chaotic scattering approach to encoding would select pixels in which to encode a given payload through a pseudo-random, repeatable algorithm. This method is not subject to the present limiting factor of the available edge sets, as all of the pixels in the image become available for encoding.

To justify our second approach in the face of PoV steganalysis and its inherent vulnerability to the Step Effect, we propose the addition of arbitrarily modified pixels, which are different from the original image but are not meaningfully altered, that is, they are not involved in the encoding of the payload. This provides an additional barrier to unauthorized access to the original payload, as attempts to detect manipulated pixels will discover many 'false

positives' that are not truly components of the encoding; lacking the original pseudo-random algorithm it then becomes much more resource-intensive to retrieve the original message, because it is impossible to discern which values are legitimate and which values are acting as red herrings. Given that our goal in a steganographic implementation continues to be effective concealment, we hold that these strategies would allow the elimination of the additional protections provided by the encryption protocol once encryption has been determined to be nearly or completely superfluous in regards to its effect on overall security. This addition would render the payload more thoroughly secured, but the resultant increased vulnerability to detection through steganalysis engenders further exploration of this matter.

Expanding base functionality

Following the potential future modifications proposed, we set forth the option of broadening the utilities offered by our application.

As a consequence of our present limited payload capacity, we were circumscribed to only allow for short text strings as potential payloads. After improving the average number of encodable payload bytes, the possibility to allow for other data or file formats as payloads begins to emerge. We assume that a typical text file will be encoded on a given operating system with fewer bytes than typical instances of other data types, such as images, audio, and video, although this is not guaranteed to be true. With the payload capacity sufficiently optimized, we would then proceed to investigate the viability of other data types as payloads; we acknowledge that this has, at this point, not been fully explored due to the preventative circumstances described, and we do expect that if other formats were to become feasible, they would likely be subject to restrictive size limits and ultimately may be determined to not be a practical use of our steganographic methods as they currently exist.

Either in combination with or as an alternative to delivering functionality for additional types of payload inputs, we present the prospect of supporting additional types of cover objects. Image formats in particular are suited for manipulation with steganographic manipulation because their primary output is visual; altering pixel values in image data can be expected to alter the final appearance of the file, but it will still indisputably be an image file. Many other file types do not contain multitudes of modifiable objects that, when altered, will not damage the integrity of the file or its expected function. In addition, any file format that undergoes lossy compression is not suitable for encoding, as the bit values will be automatically adjusted and the payload will be irretrievable; this is why we restricted our potential cover objects only to images with the *portable network graphics* (PNG) extension, which mandates lossless compression. Given these considerations, we would further explore the viability of encoding the pixels of other visual file extensions such as the *graphics interchange format* (GIF) and the *tag image file format* (TIFF). We also tentatively put forward the potential of encoding in audio file formats which do not use lossy compression, but because our exploration has only assessed pixel encoding schemes this would be a substantial undertaking.

Closing statements

In summation, our application has accomplished what it has set out to do: it provides users with an interface to encode and decode steganographic images for use in covert communications. Our two algorithms allow the user to customize their encoding to their priorities, whether they are storage and runtime efficiency or whether they are a reasonable degree of security against detection by steganalysis and the guaranteed security of the payload data, should its existence be discovered. Although both of our algorithms are imperfect, they provide a promising starting point for further exploration of steganographic algorithms that provide undetectability in the face of an adversary, security of the payload, and optimal spatial and runtime security.

REFERENCES

[1] H. Kaur and J. Rani, "A Survey on different techniques of steganography," MATEC Web of Conferences, vol. 57, p. 02003, 2016.

[2] J. Fridrich, M. Goljan, and R. Du, "Detecting LSB steganography in color, and gray-scale images," IEEE Multimedia, vol. 8, no. 4, pp. 22–28, 2001.

[3] J. Fridrich, "Steganography in Digital Media: Principles, Algorithms, and Applications. ," Cambridge University Press, 2009.

[4] "A secure image steganography algorithm based on least significant bit and integer wavelet transform," Journal of Systems Engineering and Electronics, vol. 29, no. 3, p. 639, 2018. W. C. Eastom, 2013.

[5] L. Ruiyuan and M. Jian, "Research on Improved Canny Edge Detection Algorithm," vol. 232, 2018, doi: 10.1051/matecconf/201823203053. [Online]. Available: https://macalester.on.worldcat.org/oclc/8081436040

[6] M. McBride, "Fernet system for symmetric encryption," 23-Feb-2020. [Online]. Available: https://www.pythoninformer.com/python-libraries/cryptography/fernet/#:~:text=Ciphertext%20%2D%20the%20encrypted%20version%20of,using%20the%20PKCS7%20padding%20algorithm. [Accessed: 20-Oct-2020]

[7] R. Roy, A. Sarkar, and S. Changder, "Chaos based Edge Adaptive Image Steganography," vol. 10, pp. 138–146, 2013, doi: 10.1016/j.protcy.2013.12.346. [Online]. Available: https://macalester.on.worldcat.org/oclc/5497189490

[8] Ron Crandall, "Some Notes on Steganography", Posted on Steganography Mailing List, 1998. Source: http://www.dia.unisa.it/~ads/corsosecurity/www/CORSO-0203/steganografia/LINKS%20LOCALI/matrix-encoding.pdf

[9] X. Li, B. Yang, D. Cheng, and T. Zeng, "A generalization of lsb matching", IEEE Signal Processing Letters, vol. 16, no. 2, pp. 69-72, 2009.

[10] Bin Li, Junhui He, Jiwu Huang, Yun Qing Shi, "A Survey on Image Steganography and Steganalysis", Journal of Information Hiding and Multimedia Signal Processing, Vol. 2, No. 2, April 2011, pp. 141-173.

[11] J. Díaz, "Basic Steganography and Steganalysis," 21-Apr-2015. [Online]. Available: https://www.incibe-cert.es/en/blog/basic-steganography-and-steganalysis. [Accessed: 2020]

[12] S. Chaeikar, A. Ahmadi, and P. of the 10th I. C. / C. M. and S. (ICCMS 2018), SW. ACM, 2 Penn Plaza, Suite 701, New York, NY 10121-0701, USA, 2018, p. p.14-18 [Online]. Available: https://macalester.on.worldcat.org/oclc/7856736868

[13] C.V. Serdean, M. Tomlinson, J. Wade, A.M. Ambroze, "Protecting Intellectual Rights: Digital Watermarking in the wavelet domain", IEEE Int. Workshop Trends and Recent Achievements in IT, pp. 16-18, 2002.

[14] N.Provos, P.Honeyman, "Hide and Seek: An Introduction to Steganography", IEEE Security and Privacy, Vol. 1, No. 3, 2003, pp. 32–44.

[15] Ratnakirti Roy, Suvamoy Changder, Anirban Sarkar, Narayan C Debnath, "Evaluating Image Steganography Techniques: Future Research Challenges", International Conference on Computing, Management and Telecommunications (ComManTel 2013) [IEEE], pp. 309 – 314, January 21 - 24, 2013.

[16] Li Bin, Mehdi Samiei Yeganeh, "Comparison for Image Edge Detection Algorithms", IOSR Journal of Computer Engineering, Vol. 2, Issue. 6, July-August, 2012.

[17] F. Mai, Y. Hung, H. Zhong, and W. Sze., "A hierarchical approach for fast and robust ellipse extraction", Pattern Recognition, Vol. 41, No. 8, pp.2512–2524, August 2008.

[18] Sergei Azernikov, "Sweeping solids on manifolds", Symposium on Solid and Physical Modeling,, pp.249–255, 2008.

[19] A. Pradhan, K. R. Sekhar, and G. Swain, "Adaptive PVD Steganography Using Horizontal, Vertical, and Diagonal Edges in Six-Pixel Blocks," 2017, doi: 10.1155/2017/1924618. [Online]. Available: https://macalester.on.worldcat.org/oclc/7179536498

[20] D. Lerch-Hostalot and D. Megias, "LSB matching steganalysis based on patterns of pixel differences and random embedding," 2017 [Online]. Available: https://macalester.on.worldcat.org/oclc/8660964066

[21] M. Mishra, A. Ranjan Routray, and S. Kumar, "High Security Image Steganography with Modified Arnold's Cat Map," vol. 37, no. 9, pp. 16–20, 2012, doi: 10.5120/4636-6685. [Online]. Available: https://macalester.on.worldcat.org/oclc/7378899184