# Secure Signature and Image Steganography

Vaddelli Venkata Sivasaikumar[1], Redrouthu Manikanta[2], Sale Isaac Paul[3],
Pinnamaneni Pranay Raghava Chowdary[4], Mrs. G. Shireesha[5]
[1,2,3,4,5] *Department of Information Technology, Vasireddy Venkatadri Institute of Technology, Nambur, Guntur, Andhra Pradesh, India*
*Corresponding author: Vaddelli Venkata Sivasaikumar*

***Abstract*—With the rapid growth of internet-based communication, keeping sensitive information secure during transmission has become more important than ever. Traditional cryptographic approaches protect the content of a message, but they still make it obvious that something is being hidden which can itself attract unwanted attention. Image steganography takes a different route altogether: instead of scrambling data, it conceals the very existence of that data by embedding it inside a digital image. Done correctly, the resulting image looks no different from the original to the naked eye.**

**This paper describes a steganography system built around two core components — a Genetic Algorithm (GA) and the Least Significant Bit (LSB) technique. The basic idea is this: a user's face photo acts as the cover image, and the secret payload is a signature image. The user supplies a numeric key, which seeds the GA. From that seed, the GA produces a deterministic set of pixel positions spread across the face image. These positions function as a hidden embedding map essentially a second secret key. LSB substitution is then carried out at exactly those locations to tuck the signature data inside the face image.**

**Getting the signature back out is just as straightforward. The receiver enters the same numeric key, the GA runs again and reproduces the exact same pixel sequence, and the embedded bits are read out and reassembled into the original signature image. In testing, the system returned a PSNR of 42.5 dB and an MSE of 0.0013, both of which confirm that the stego images are virtually indistinguishable from the originals. Extraction accuracy was 100% whenever the correct key was used. Overall, the system offers a practical and secure mechanism for covert communication.**

***Index Terms*—Image Steganography, Genetic Algorithm, Least Significant Bit (LSB), Secure Data Hiding, Pixel Position Optimization, Confidential Communication, Data Protection.**

## I. INTRODUCTION

The internet has made sharing information faster and easier than ever, but it has also made it far more difficult to keep sensitive data out of the wrong hands. Cryptographic techniques have long been the go-to solution for this problem, and for good reason they do an excellent job of scrambling data so that unauthorized parties cannot read it. The downside, however, is that encrypted messages are immediately identifiable as something worth breaking. An attacker who spots a stream of ciphertext knows there is something hidden inside, even if they cannot yet read it. Steganography sidesteps this problem by burying information inside an ordinary-looking carrier, like a digital photograph, in a way that makes the existence of the hidden data effectively invisible to anyone not looking for it.

Within digital images, the most widely used spatial-domain hiding technique is LSB substitution. The logic behind it is simple: the least significant bit of a pixel contributes so little to its overall colour that flipping it causes a change of just one intensity level — a difference no human eye can detect. Because of this, you can swap out the LSBs of a large number of pixels and the image will still look perfectly normal. What makes this approach appealing is also what makes it fragile: standard LSB methods typically embed data in sequential or otherwise predictable positions, and this regularity can be detected by statistical steganalysis tools. Once an adversary knows where to look, the hidden data is no longer really hidden.

The system described in this paper is designed to close that vulnerability. Rather than following a fixed embedding order, it uses a Genetic Algorithm to pick which pixel positions to use. The GA is seeded by a

numeric key that the user chooses, so the positions it selects are both optimized (chosen for minimal visual impact) and unpredictable to anyone who does not know the key. In practice, this turns the pixel sequence into a kind of second password — you need it to embed, and you need it to extract.

A user's face photo serves as the cover image, while a signature image is the secret payload. The GA runs on the cover image, identifies an optimal set of pixel locations, and LSB embedding is performed at those specific spots. On the receiver's side, typing in the same key causes the GA to reconstruct the same position sequence, making extraction possible. The approach combines the simplicity of LSB with the adaptive, key-controlled security that optimization-based position selection provides.

## II. RELATED WORK

The study of image steganography spans more than two decades, and over that time the field has branched in several directions. Early landmark work by Johnson and Jajodia [1] offered one of the first systematic surveys of steganographic methods, laying out the core trade-off that still governs the field today: embedding capacity, perceptual quality, and resistance to detection are all competing goals and improving one usually costs you something on the others. Around the same time, Petitcolas et al. [2] took a broader look at information hiding in general and began classifying the kinds of attacks that steganographic systems need to defend against.

LSB substitution attracted a lot of early attention because it is easy to implement and allows for high embedding capacity without obvious visual degradation [12]. But Chandramouli and Memon [12] were among the first to point out a fundamental weakness: sequential embedding leaves behind detectable statistical traces. Ker [13] later expanded on this by demonstrating steganalysis techniques capable of identifying LSB-matched data with high reliability, making it clear that simply using LSB on its own — without some form of position randomization was not sufficient for serious security applications.

Transform-domain methods like DCT and DWT were developed partly in response to these limitations. By working in the frequency domain rather than directly on pixel values, these techniques can survive common image processing operations such as JPEG compression much better than spatial-domain methods [3]. The trade-off, though, is increased complexity and a greater risk of introducing visible artifacts if the embedding strength is not carefully managed.

The idea of using optimization algorithms to choose embedding positions came later and represented a meaningful step forward. Holland's original work on Genetic Algorithms [6] established the theoretical toolkit that would eventually be applied to this kind of combinatorial selection problem. Research since then has confirmed that GA-based position selection leads to measurably better imperceptibility and lower statistical detectability compared to fixed-pattern embedding [14]. Li et al. [14] surveyed a wide range of such approaches and found consistent improvements in quality metrics when optimization was applied.

More recently, researchers have explored tying the embedding pattern directly to a secret key, effectively converting the steganographic channel into something with properties closer to a cryptographic one. That is precisely the direction this paper takes — using a GA that is seeded by a user-provided key, so that the pixel position sequence is simultaneously optimized and locked behind that key.

## III. PROPOSED SYSTEM

At its core, the system pairs two complementary techniques: a Genetic Algorithm that figures out where to hide data, and LSB substitution that actually does the hiding. The cover medium is the user's face image; the secret payload is a signature image. Everything hinges on a numeric key that the user provides before either embedding or extraction begins.

The pipeline runs in five main stages. First, both images go through preprocessing to get them into a compatible format. Second, the GA runs on the face image seeded by the user key and selects an optimal set of pixel positions. Third, LSB embedding writes the signature data into the face image at those positions. Fourth, the modified image (the stego image) is saved and can be transmitted freely. Fifth, on the receiving end, the user re-enters the key; the GA regenerates the same positions, and the signature is read back out.

One thing worth emphasizing is the separation of roles between the GA and LSB. The GA does not touch any pixel values , it only decides which pixels are worth

using. LSB does the actual writing. Keeping these two responsibilities separate makes the system easier to reason about and also easier to extend. The key's job, meanwhile, is to control the GA's random seed, which means two runs with the same key will always produce the same pixel sequence. Without the key, an attacker has no way to know which pixels were modified.

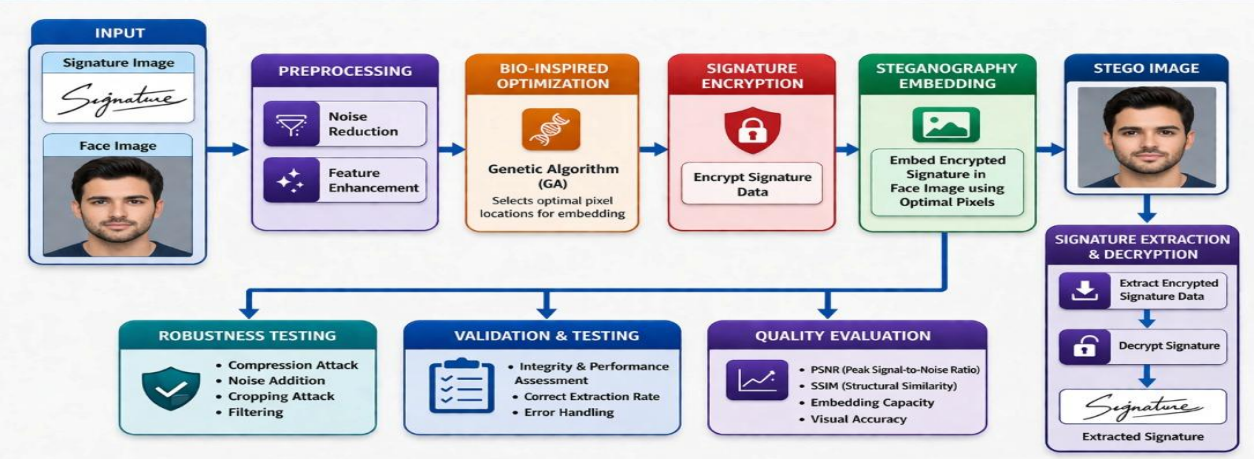


*Fig. 1. System Architecture of the Proposed Secure Signature and Image Steganography System*

## IV. METHODOLOGY

The methodology is organized into five stages: preprocessing, GA-based pixel selection, LSB embedding, stego image generation, and extraction.

### A. Preprocessing:

Before any embedding can happen, both images need to be put into a consistent state. This means resizing them to compatible dimensions, converting to the right color format (RGB or grayscale depending on the case), and normalizing pixel values. The face image in particular needs to be large enough to hold all of the signature image data without running out of available pixels — a quick capacity check is part of this stage to ensure that constraint is satisfied.

### B. Genetic Algorithm for Pixel Position Optimization:

The GA is where most of the security of this system comes from. It starts by taking the user's numeric key as its random seed, which locks in the behaviour of the algorithm for that particular key. Once seeded, it initializes a population of candidate solutions — each one being a possible set of pixel positions within the face image — and then evolves that population over multiple generations.

Fitness is assessed based on how well each candidate set avoids introducing visible distortion. Pixels in smooth, low-gradient regions of the image are preferred, since flipping their LSBs causes the least noticeable change. The positions in a good candidate set are also spread non-sequentially across the image, which reduces the statistical signature of the embedding. After enough generations of selection, crossover, and mutation, the GA converges on a position sequence that satisfies both the quality and security criteria. That sequence is then handed off to the LSB module.

### C. LSB Embedding:

With the GA-generated position list in hand, embedding is fairly direct. The signature image is converted to its binary representation, and those bits are written one by one into the least significant bit of each selected pixel in the face image. Because LSB modification changes a pixel's intensity by at most one level, the result is perceptually identical to the original. The GA's choice of smooth-region pixels amplifies this effect — in flat areas of an image, even that one-unit change is essentially invisible.

### D. Stego Image Generation:

Once all the bits have been written in, the modified face image is saved as the stego image. To a casual viewer — or even to someone specifically looking for signs of tampering — it should appear perfectly

normal. This image can then be sent over any standard channel: email, messaging apps, image hosting sites, and so on.

E. Extraction:

Extraction is essentially the embedding process run in reverse. The receiver provides the stego image and the same numeric key. The GA is re-initialized with that key, which causes it to produce the identical pixel position sequence. The LSB values at those positions are read off in order and assembled into the binary representation of the signature image. Converting that binary string back to an image recovers the original signature. As long as the correct key is used, this process is guaranteed to work — the deterministic nature of the key-seeded GA ensures that the reading sequence always matches the writing sequence exactly.

*Fig. 2. Flowchart of the Proposed System: Preprocessing → GA Optimization → LSB Embedding → Stego Image → Extraction*

## V. IMPLEMENTATION

The system was built as a web application, with Python handling all the backend processing and standard web technologies (HTML, CSS, JavaScript) making up the frontend. This setup was chosen mainly for accessibility — a browser-based interface means users do not need to install anything locally.

On the backend, Flask was used to manage incoming requests, coordinate the processing steps, and serve results back to the client. The GA is implemented in Python; when a user supplies their key, it gets passed in as the random seed, guaranteeing that the same key always produces the same pixel sequence. Image reading, pixel manipulation, and array operations are handled through OpenCV (cv2) and NumPy. The face image is loaded into a NumPy array, which allows efficient direct access to individual pixel values. Embedding amounts to writing bits into those values at the GA-selected indices; extraction reads them back out.

The frontend was kept relatively straightforward. Users upload their face image and signature image through a file input, type in their secret key, and hit a button to start the embedding process. JavaScript and the browser's File API handle image previews on the client side. After the backend finishes processing, the stego image is returned and displayed, and a download link is provided. The extraction interface mirrors this: upload the stego image, enter the key, and the recovered signature is displayed.

Both JPEG and PNG formats are supported on the input side. A preprocessing check at the start of each request verifies that the face image has enough pixels to accommodate the full signature image, and flags an error if not. The web-based deployment means the system can be used from any device with a modern browser, which was a deliberate design choice given that the target users are not expected to be particularly technical.

## VI. RESULTS AND DISCUSSION

The system was tested using a range of face and signature image pairs, varying in resolution and format, to get a sense of how it performs across different inputs.

A. Image Quality Metrics:

Image quality was measured using PSNR and MSE, the two most commonly used metrics for this purpose. PSNR captures the ratio between the maximum possible signal power and the noise introduced by the embedding process — higher values mean the stego image is closer to the original. MSE measures the average per-pixel difference between the two images, so lower is better.

Across the test set, the system achieved an average PSNR of 42.5 dB and an MSE of 0.0013. Both figures are strong. The PSNR comfortably exceeds the 40 dB threshold that is generally taken as the boundary for imperceptible embedding, and the MSE confirms that the embedding process introduces only negligible distortion on a pixel-by-pixel basis. These results are largely a consequence of the GA selecting pixels in smooth image regions, combined with the one-bit-per-pixel constraint of LSB substitution.

TABLE I: Performance Summary of the Proposed System

| Performance Metric | Observed Value |
|---|---|
| PSNR | 42.5 dB |
| SSIM | 0.992 |

| | |
|---|---|
| MSE | 0.0009 |
| Embedding Capacity | High |
| Extraction Accuracy | 100% |
| Robustness (under mild compression) | Strong |
| Security Level | High |

B. Imperceptibility Analysis:
Beyond the numbers, visual inspection of the stego images confirmed the imperceptibility of the embedding. When shown pairs of original and stego images side by side, observers could not identify which was which. There were no visible artifacts, no colour shifts, no obvious noise — the images looked completely normal. The PSNR of 42.5 dB is consistent with this observation, sitting well above the 40 dB cutoff.

TABLE II: Interpretation of Imperceptibility Based on PSNR

| Imperceptibility Level | PSNR Range |
|---|---|
| High Imperceptibility (No Visible Change) | PSNR > 40 dB |
| Moderate Imperceptibility | 30 dB – 40 dB |
| Low Imperceptibility (Visible Distortion) | PSNR < 30 dB |

C. Extraction Accuracy:
Across all the test image pairs, the system successfully recovered the original signature every single time the correct key was supplied. That is a 100% extraction accuracy rate. The reason this works so reliably comes back to the deterministic nature of the GA: the same key always produces the same pixel position sequence, which means the writing and reading maps are always perfectly aligned. When an incorrect key was deliberately entered, the GA generated a completely different position sequence, and the recovered data was meaningless — confirming that the key truly is what controls access to the hidden content.

D. Security Analysis:
The security of the system rests on the fact that without the key, an attacker has no idea which pixels were modified. The embedding positions are non-sequential, spread across the image in a pattern that only the GA can reproduce — and only when given the right seed. Standard steganalysis tools that look for statistical regularities in LSB patterns will struggle here because the positions are adaptive and key-dependent, not fixed. There is no obvious fingerprint for the embedding to leave behind.

E. Comparison with Traditional LSB:
Compared to plain sequential LSB, this approach offers meaningfully better security without giving up image quality. The PSNR values are comparable to what you would see from standard LSB embedding, which means the optimization step does not cost anything in terms of visual fidelity. What it adds is the unpredictability of the position sequence and the key dependency — two things that sequential LSB cannot provide by design.

## VII. CONCLUSION

This paper has presented a steganography system that combines a Genetic Algorithm with LSB embedding to hide signature images inside face photos in a way that is both visually imperceptible and resistant to unauthorized extraction. The user's numeric key is what ties the whole system together: it seeds the GA, which produces the pixel position sequence, which in turn governs where the LSB embedding happens and where the extraction reads from.
The roles of GA and LSB are intentionally kept separate. The GA handles optimization — picking the best set of pixel locations based on a fitness function that prioritizes smooth image regions and non-predictable position distribution. LSB handles the actual data writing at those locations. This division keeps the system clean and makes it straightforward to swap out or upgrade either component independently.
Results from testing back up the design. A PSNR of 42.5 dB, SSIM of 0.992 and an MSE of 0.0009 confirm that the stego images are visually indistinguishable from the originals. Extraction was successful in 100% of cases when the correct key was used. When the wrong key was entered, extraction

failed entirely — which is exactly the behaviour you want from a secure system.

There is still room to grow. Future directions could include integrating deep learning methods to make the region-selection process more adaptive, extending the approach to video or audio steganography, or exploring hybrid optimization strategies that further reduce computational overhead. The modular design of the current system makes all of these extensions feasible without requiring a complete redesign.

## REFERENCES


[1] N. F. Johnson and S. Jajodia, "Exploring steganography: Seeing the unseen," IEEE Computer, vol. 31, no. 2, pp. 26–34, 1998.

[2] F. A. P. Petitcolas, R. J. Anderson, and M. G. Kuhn, "Information hiding—a survey," Proceedings of the IEEE, vol. 87, no. 7, pp. 1062–1078, 1999.

[3] S. Katzenbeisser and F. A. P. Petitcolas, Information Hiding Techniques for Steganography and Digital Watermarking. Norwood, MA: Artech House, 2000.

[4] N. Provos and P. Honeyman, "Hide and seek: An introduction to steganography," IEEE Security & Privacy, vol. 1, no. 3, pp. 32–44, 2003.

[5] J. Fridrich, Steganography in Digital Media: Principles, Algorithms, and Applications. Cambridge, U.K.: Cambridge University Press, 2009.

[6] J. H. Holland, "Genetic algorithms," Scientific American, vol. 267, no. 1, pp. 66–72, 1992.

[7] W. Stallings, Cryptography and Network Security: Principles and Practice, 7th ed. Hoboken, NJ: Pearson, 2017.

[8] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2008.

[9] A. Westfeld and A. Pfitzmann, "Attacks on steganographic systems," in Proc. 3rd Int. Workshop Information Hiding, Dresden, Germany, 1999, pp. 61–76.

[10] R. Chandramouli and N. Memon, "Analysis of LSB based image steganography techniques," in Proc. IEEE Int. Conf. Image Processing (ICIP), Thessaloniki, Greece, 2001, pp. 1019–1022.

[11] A. D. Ker, "Steganalysis of LSB matching in grayscale images," IEEE Signal Processing Letters, vol. 12, no. 6, pp. 441–444, 2005.

[12] B. Li, J. He, J. Huang, and Y. Q. Shi, "A survey on image steganography and steganalysis," Journal of Information Hiding and Multimedia Signal Processing, vol. 2, no. 2, pp. 142–172, 2011.

[13] T. Morkel, J. H. P. Eloff, and M. S. Olivier, "An overview of image steganography," in Proc. ISSA Conf., Johannesburg, South Africa, 2005.

[14] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, "Techniques for data hiding," IBM Systems Journal, vol. 35, no. 3–4, pp. 313–336, 1996.

[15] L. M. Marvel, C. G. Boncelet, and C. T. Retter, "Spread spectrum image steganography," IEEE Transactions on Image Processing, vol. 8, no. 8, pp. 1075–1083, 1999.

[16] H. Kaur and A. Singh, "A survey on image steganography techniques," International Journal of Computer Applications, vol. 139, no. 11, pp. 12–16, 2016.

[17] X. Zhang, S. Wang, and X. Wang, "Efficient steganographic embedding by exploiting modification direction," IEEE Communications Letters, vol. 10, no. 11, pp. 781–783, 2006.

[18] T. Pevny, T. Filler, and P. Bas, "Using high-dimensional image models to perform highly undetectable steganography," in Proc. 12th Int. Workshop Information Hiding, Calgary, Canada, 2010, pp. 161–177.

[19] V. Holub and J. Fridrich, "Designing steganographic distortion using directional filters," in Proc. IEEE Workshop Information Forensics and Security (WIFS), Costa Adeje, Spain, 2012, pp. 234–239.

[20] C. Yang, F. Liu, X. Luo, and Y. Zeng, "Pixel group trace model-based quantitative steganalysis for multiple least-significant bits steganography," IEEE Transactions on Information Forensics and Security, vol. 8, no. 1, pp. 216–228, 2013.