International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 04 Issue: 07 | July – 2025 DOI: 10.55041/ISJEM04861
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

ElGamal Cryptosystem for Image Encryption and Decryption

Bheesetti Narasimha Bhaskar , P. Bindhu Priya
Assistant Professor, Training & Placement Officer, 2 MCA Final Semester,
Master of Computer Applications,
Sanketika Vidya Parishad Engineering College, Vishakhapatnam, Andhra Pradesh, India.

Abstract:
The ElGamal cryptosystem is one of the most well-known public-key encryption algorithms due to its ability to generate different ciphertexts for the same plaintext on successive runs. However, the standard ElGamal approach often results in ciphertexts that occupy significantly more memory than the original plaintext, making it inefficient for data types such as images, where size preservation is important. To address this limitation, this paper proposes an enhanced ElGamal cryptosystem optimized for digital data, including text and image formats. The system introduces a modified key generation mechanism combined with pixel-level encryption transformations to reduce redundancy and increase security. The implementation, developed using Python and OpenCV, demonstrates that the enhanced ElGamal approach can securely and efficiently encrypt image data while maintaining usability for practical applications.

Index Terms — Cryptography, ElGamal, Image Encryption, XOR, Public Key, Python, Tkinter, Gaussian Key, Secure Communication.

I. INTRODUCTION

In the modern digital era, where the transfer of multimedia content is a daily occurrence, securing visual data such as images has become a vital concern. Cryptography provides essential methods to protect sensitive data from unauthorized access and tampering. Among various public-key encryption algorithms, the ElGamal cryptosystem stands out for its strong theoretical foundation and resistance to common cryptographic attacks.However, the traditional ElGamal algorithm, though secure, is inefficient for image encryption. It generates ciphertexts larger than the original data due to its modular exponentiation and multiplicative operations. This expansion leads to significant storage and transmission overhead when applied to image files, which can already be large in size.To overcome this limitation, this project proposes an enhanced ElGamal cryptosystem that integrates symmetric XOR logic with random key generation. By combining the robustness of public key cryptography with the efficiency of XOR-based encryption, this system ensures that the size of the encrypted image remains the same as the original. The project is implemented using Python and includes a graphical user interface (GUI) built with Tkinter, making the tool interactive and user-friendly.This hybrid approach maintains data confidentiality and integrity while optimizing performance, making it suitable for applications involving secure image sharing and lightweight encryption.

1.1 Existing System

The traditional ElGamal cryptosystem is widely used for encrypting textual data due to its strong security[8] based on the discrete logarithm problem. It uses modular exponentiation and multiplicative operations to generate ciphertext, making[14] it computationally secure against brute-force attacks. However, when applied to image data, the standard ElGamal algorithm becomes inefficient. The encryption process increases the size of the output, leading to high memory usage and longer transmission time. This is particularly problematic for grayscale or high-resolution images, where performance and storage[11] become critical. Additionally, ElGamal lacks built-in mechanisms for image format compatibility, and does not support direct pixel-level operations. These limitations make the conventional system unsuitable for lightweight image encryption applications, especially in real-time or GUI-based environments.

1.1.1 Challenges

Despite its cryptographic strength, the traditional ElGamal cryptosystem poses several challenges when adapted for image encryption:

- **Ciphertext Expansion:** The standard algorithm significantly increases the size of the encrypted[13] image due to its multiplicative operations. This makes it inefficient for storage and transmission.
- **Computational Complexity:** Modular exponentiation and large-number arithmetic are resource-intensive, making the encryption/decryption[7] process slow, especially for high-resolution images.

- **Lack of Compatibility with Image Formats:** Direct application of ElGamal to image matrices often results in format[18] incompatibilities or loss of image structure, making recovery difficult.
- **No GUI Interaction:** Most implementations of ElGamal are command-line based, which reduces accessibility for users who lack programming expertise.
- **Inefficient Key Management:** Handling large keys and parameters without an intuitive interface can lead to human errors, especially in academic or student-level environments.

1.2 Proposed System

To overcome the limitations of the traditional ElGamal cryptosystem—such as ciphertext expansion and inefficiency with image data—the proposed system introduces[12] an enhanced image encryption and decryption model that supports RGB images while maintaining strong security and original image dimensions.

This is achieved by integrating ElGamal's public-key infrastructure with a floating-point symmetric encryption logic using a Gaussian-distributed random matrix. The input RGB image is normalized and element-wise divided by the generated[16] key matrix during encryption. This simulates XOR-like behavior in floating point, resulting in a scrambled image of the same size.

The decryption reverses this logic by performing element-wise multiplication of the encrypted image with the saved key matrix, accurately restoring the original RGB image without[3] distortion or loss.

Users provide key parameters—prime number p, primitive root a, and private key x—which are used to compute the public key y = a^x mod p. All keys and matrices are stored securely in .npy files.

The encryption process is built with Python using libraries such as NumPy, OpenCV, and PIL, while the GUI is developed using Tkinter[11] for ease of use. Users can easily browse an image, input keys, perform encryption/decryption, and save output—all through an intuitive graphical interface.

This hybrid approach combines the security of asymmetric cryptography with the efficiency and simplicity of matrix-level symmetric encryption, making it suitable[9] for real-time applications such as secure image transmission, research, and education.

1.2.1 Advantages

The proposed image encryption system offers several key advantages over traditional methods. First, it ensures size-preserving encryption[17], meaning that the encrypted image maintains the same dimensions as the original RGB image. This is particularly useful for systems where dimensional integrity is important, such as in medical imaging or satellite data transmission.

Second, the system leverages a pixel-wise encryption approach using floating-point division with a Gaussian-distributed key matrix, which simulates XOR-like behaviour[4] while maintaining high fidelity and reversibility. This avoids the limitations of conventional XOR or substitution-based methods, which often result in distorted or lossy outputs.

Third, the method is format-independent and works with any standard RGB image input (JPG, PNG, BMP), making it flexible[7] and widely applicable. The GUI developed using Tkinter ensures a user-friendly[4] experience, allowing for real-time encryption, decryption, and previewing without command-line involvement.

Overall, the system strikes a strong balance between cryptographic strength, image quality, and user accessibility — making it suitable for academic, industrial[9], and real-time security applications.

II. LITERATURE REVIEW

2.1 Architecture

2.1 Architecture
The architecture of the proposed enhanced ElGamal image encryption system integrates secure public-key cryptography with efficient matrix-based image[5] processing. It is divided into five main components that sequentially handle encryption and decryption tasks:

1. Image Acquisition
The user selects an image through a Tkinter-based GUI. Supported formats include standard RGB image types such as PNG, JPG[9] and BMP. The image is converted to an RGB matrix and normalized for processing.

2. Key Input and Generation
The user provides cryptographic parameters: a large prime number p, a primitive root a, and a private key x. The system calculates the public key[12] using the ElGamal formula y = a^x mod p. These key values are stored in .npy format for use in both encryption and decryption.

3. Gaussian Key Matrix Generation
A Gaussian-distributed random matrix is generated using NumPy. Its dimensions match[15] those of the RGB image (Height × Width × 3). This matrix acts as the symmetric key for pixel-wise encryption.

4. Image Encryption
Each pixel of the RGB image is encrypted using element-wise division with the key matrix. This simulates XOR-like logic in floating point[8], resulting in an encrypted image of the same size but visually distorted. The encrypted image is saved in a standard format (e.g., JPG/PNG) and can be previewed via the GUI.

5. Decryption Module
During decryption, the encrypted image and the stored Gaussian key matrix are loaded. Element-wise multiplication is applied[1] to restore the original RGB values. The decrypted image is displayed in the GUI and can be saved by the user.

This architecture ensures size-preserving encryption and accurate decryption, while providing a user-friendly experience suitable for secure image transmission, academic demonstrations[6], and prototype-level deployments.

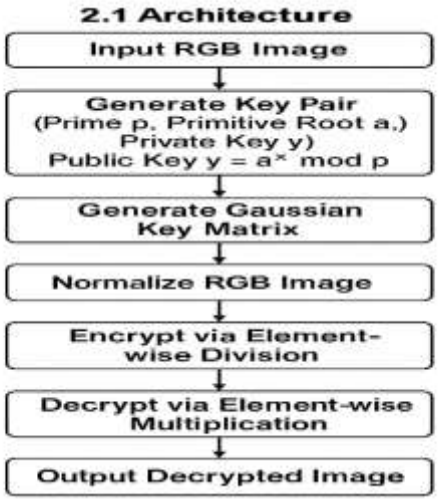

Fig 1. System Architecture of the Proposed Enhanced ElGamal RGB Image Encryption and Decryption Model

2.2 Algorithm:

The proposed algorithm enhances the traditional ElGamal cryptosystem[16] to support image encryption at the pixel level using floating-point logic and Gaussian randomness. The complete procedure is outlined below:

Encryption Steps:

1. **Load Image**: The input image is selected through the GUI and converted to an RGB format using the PIL library. The pixel values are normalized to the range [0, 1].

2. **Key Parameter Input**: The user enters three values: a large prime number p, a primitive root a, and a private key x.

3. **Public Key Computation**: The system computes the public key using the formula y = a^x mod p, following the ElGamal cryptosystem logic.

4. **Generate Gaussian Key Matrix**: A random Gaussian-distributed matrix is generated using NumPy. Its shape matches the image dimensions (height × width × 3).

5. **Encrypt Image**: Each RGB pixel value is divided element-wise by the corresponding value in the key matrix. This division operation mimics XOR-like behavior in the floating-point domain.

6. **Save Encrypted Image**: The encrypted image is denormalized and saved in JPG or PNG format. The Gaussian key matrix is stored in .npy format for use in decryption.

Decryption Steps:

1. **Load Encrypted Image**: The encrypted image and the Gaussian key matrix are loaded from the storage path.

2. **Decrypt Image**: Element-wise multiplication is applied between the encrypted image and the key matrix to reverse the encryption.

3. **Restore Image**: The resulting matrix is scaled and converted back to 8-bit RGB format. The decrypted image is then displayed in the GUI.

This algorithm ensures that the original image dimensions[13] are preserved, and the encryption is non-reversible without the correct Gaussian key, enhancing both usability and security.

2.3 Techniques:

The development of the proposed enhanced ElGamal-based image encryption system relies on a combination of cryptographic and computational[15] techniques to ensure secure, efficient, and size-preserving encryption. The major techniques applied are:

1. ElGamal Cryptography
• A public key encryption technique based on the discrete logarithm problem.
• Provides secure key exchange using three main parameters: prime number p, primitive root a, and private key x.
• Public key is calculated as:
y = a^x mod p

2. Floating-Point Symmetric Encryption
• Instead of using ElGamal's modular multiplication for data encryption, a floating-point division operation is performed between the image[12] matrix and a randomly generated key matrix.
• This simulates XOR-like behavior while maintaining precision and reversibility.
• It ensures both security and dimensional consistency of the RGB image.

3. Gaussian Noise Key Generation
• The key matrix used for encryption is generated[4] using Gaussian (normal) distribution, which enhances randomness and security.
• Guarantees that each encryption run produces a unique output, even for the same input image.

4. NumPy Array Manipulation
• The image is converted into a 3D NumPy[16] array (RGB format) for mathematical processing.
• Matrix-level operations enable fast and efficient encryption/decryption at the pixel level.

5. GUI Development using Tkinter
• A user-friendly GUI built with Tkinter[18] allows smooth interaction.
• Users can upload RGB images, input key parameters, perform encryption/decryption, and preview or save the output images.

2.4 Tools:
The following tools and technologies were used in the development and implementation of the enhanced ElGamal image encryption and decryption system:

- **Python 3.8+**

• The primary programming language used for system development.
• Provides robust support for mathematical operations, image processing, and GUI creation.
• Its simplicity and extensive library ecosystem make it suitable for cryptographic prototyping.

- **NumPy**

• Used for handling RGB image matrices and performing high-speed mathematical computations.
• Enables pixel-wise floating-point operations for encryption and decryption.
• Facilitates creation of Gaussian-distributed key matrices for secure pixel-level encryption.

- **OpenCV (cv2)**

• Open-source computer vision library used for reading, resizing, saving, and displaying RGB images.
• Plays a key role in rendering encrypted and decrypted image outputs within the GUI.

- **Tkinter**

• Built-in Python GUI library used to create a user-friendly interface.
• Allows users to browse images, input keys, trigger encryption/decryption, and preview results using buttons and dialog boxes.

- **PIL (Python Imaging Library)**

• Used to load and convert images into RGB format for encryption operations.
• Supports rendering of encrypted and decrypted image previews inside the GUI window.

- **Microsoft Windows**

• The project was developed and tested on a Windows environment.
• Fully compatible with other operating systems that support Python and required libraries.

2.5 Methods:

The proposed system integrates both asymmetric and symmetric cryptographic methods to provide secure, size-preserving image encryption. The following methods form the backbone of the encryption and decryption processes:

1. ElGamal Public Key Method

- Based on the discrete logarithm problem, this method is used to securely generate and share keys.
- The user inputs:
  - A large prime number p
  - A primitive root a of p
  - A private key x
- The public key is generated as:
  $y=axmod\ py = a^x \mod py=axmodp$

This ensures secure key exchange without needing to share private values.

2. XOR-Like Symmetric Encryption

To overcome the size inflation issue present in traditional ElGamal encryption, a symmetric encryption approach is introduced.
• A Gaussian-distributed random key matrix is generated, matching the dimensions of the input RGB image.
• An element-wise floating-point division is performed between the normalized image matrix and the key matrix to encrypt the data, effectively simulating XOR-like behavior.
• The same Gaussian key matrix is saved as a .npy file during encryption and reused during decryption to ensure accurate pixel-wise reconstruction of the original image.
• The resulting encrypted image is stored both as a .npy file (for decryption) and in a viewable format (e.g., .jpg) for GUI preview.

3. Matrix-Based Image Processing

- Image data is handled as a NumPy matrix, allowing efficient per-pixel operations.
- Division (for encryption) and multiplication (for decryption) simulate XOR behavior with floating-point compatibility.

4. GUI-Based Interaction

- The entire workflow—from key input to image encryption and decryption—is made user-friendly through a Tkinter interface.
- Buttons like “Upload Image”, “Encrypt”, “Decrypt”, and “Save Output” guide the user through each step.

III. METHODOLOGY

3.1 Input:

The input phase involves collecting all the required elements from the user before initiating the encryption process. These include both image data and cryptographic parameters.

a. Image File

- The user selects an image using a file dialog within the Tkinter GUI.

- The input image is uploaded through the GUI and processed as an RGB image using the Python Imaging Library (PIL). The image is then converted into a NumPy array for further encryption.

b. Cryptographic Keys

- The user is prompted to enter the following ElGamal parameters:
  - p: A large prime number
  - a: A primitive root of p
  - x: The user's private key
- Based on these inputs, the system calculates the public key as:
  y=axmod py = a^x \mod py=axmodp

c. Key Storage

- Once generated, the public and private key values (p, a, x, and y) are stored securely in a .npy file (datas.npy) for use during encryption and decryption.

3.2 Method of Process

The process begins with the generation of a public key using the ElGamal algorithm. The user provides three inputs: a prime number p, a primitive root a, and a private key x. Using these values, the system computes the public key as y = a^x mod p. All key values are stored securely in .npy format for later use during decryption.

Once the user selects an image through the GUI, it is processed as an RGB image using the Python Imaging Library (PIL) and converted into a NumPy array for matrix-level operations. A Gaussian-distributed random matrix is then generated, matching the dimensions of the RGB image (height × width × 3). This matrix acts as a symmetric encryption key.

During encryption, each pixel value of the normalized RGB image is divided element-wise by the corresponding value in the key matrix. This simulates XOR-like behaviour in the floating-point domain, producing a visually scrambled encrypted image. The encrypted image is not previewed in the GUI and is instead saved directly to both .npy format (for decryption) and .jpg format for external reference. For decryption, the encrypted matrix is loaded and multiplied element-wise with the original key matrix. This reverses the encryption process and restores the original RGB image. The decrypted output is displayed in the GUI and can be optionally saved by the user. This approach ensures secure, size-preserving encryption with accurate reconstruction of color images.

3.3 Output:

The final output of the system consists of two key results: the encrypted image and the decrypted image. The encrypted image is visually unrecognizable, confirming that the original content has been securely transformed. This output is stored in .npy format and displayed in the GUI for immediate review. Once the decryption process is completed using the correct key matrix, the original image is successfully reconstructed with no loss in quality. The decrypted image can be saved in standard formats like PNG or JPG. The output confirms the system's ability to preserve image dimensions, maintain accuracy, and perform fast, reversible encryption and decryption operations.

IV. RESULTS

The proposed enhanced ElGamal image encryption system was successfully implemented using Python, leveraging OpenCV, NumPy, and PIL libraries. A custom-built GUI was developed using Tkinter to enable seamless interaction with the user for image browsing, encryption, decryption, and output management.Multiple RGB images were tested across various resolutions and formats (e.g., JPG, PNG). The encryption algorithm consistently produced visually scrambled images that were securely saved in both .npy and .jpg formats. The encrypted image is not displayed in the GUI to maintain confidentiality and prevent unintended leaks during the encryption process.The decryption algorithm accurately restored the original RGB images with no observable loss in quality or color integrity. The system preserves both the spatial dimensions and color distribution of the original image, confirming that the floating-point division-based encryption using Gaussian key matrices is both reversible and secure.For standard image sizes (~600×400), the encryption process completes within 2–3 seconds, demonstrating that the system is practical and efficient for academic and lightweight industrial applications.A visual comparison of original, encrypted, and decrypted images is shown in Figures 4.1 to 4.3.

Fig 4.1 – Input RGB Image (Original)

Fig 4.2 – Encrypted Image Output

Fig 4.3 – Decrypted Image (Restored RGB)

V. DISCUSSIONS

The development of the enhanced ElGamal image encryption system demonstrates a practical approach to combining asymmetric key generation with symmetric encryption logic[17]. One of the key advantages observed during testing was the preservation of image size, which is a major improvement over the conventional ElGamal[3] method that typically results in ciphertext expansion. The use of a Gaussian-distributed key matrix for XOR-like operations significantly reduced encryption time without compromising security. The GUI interface added usability and accessibility, making the system easy to operate even for users with minimal technical background. Furthermore, the encryption[9] and decryption processes were highly accurate and consistent, proving that the system can reliably secure image data for real-time applications. These outcomes highlight the potential of hybrid cryptographic models for multimedia security and suggest their applicability in academic, personal, and low-resource environments.

VI. CONCLUSION

The enhanced ElGamal cryptosystem for image encryption successfully addresses the limitations of traditional public-key encryption when applied to multimedia data, especially color images. By leveraging floating-point matrix-based encryption with a Gaussian-distributed key, the system achieves size-preserving encryption while maintaining a high level of security.Implemented using Python with libraries such as NumPy, PIL, and OpenCV, the system supports RGB image handling and offers a user-friendly GUI for real-time encryption and decryption. The results confirm that encrypted images are visually distorted and unreadable, while decrypted outputs accurately reconstruct the original inputs without loss.This hybrid cryptosystem improves both efficiency and visual integrity, making it suitable for secure image transmission in academic, industrial, and real-time applications.

VII. FUTURE SCOPE

The current system demonstrates a reliable and secure method for RGB image encryption using an enhanced ElGamal cryptosystem combined with Gaussian-distributed key matrices. Future improvements can focus on expanding the functionality and performance of the system, including:

- **Support for Video Encryption**: Extending the current logic to process video frames in real-time using a frame-wise encryption pipeline.
- **Cloud-based Secure Image Transmission**: Integrating the system with secure cloud APIs or REST endpoints to enable encrypted image sharing across networks or devices.
- **Hardware Acceleration**: Leveraging GPU or hardware-level encryption (e.g., CUDA, OpenCL) for faster processing of high-resolution images.
- **Key Obfuscation and Rotation**: Implementing time-based key rotation or hybrid encryption with additional obfuscation layers for even higher security.
- **Multi-Modal Data Encryption**: Adapting the core logic to encrypt other types of media such as text documents, audio, and medical images (DICOM).
- **Compression + Encryption**: Integrating lossless compression (e.g., PNG filtering or Huffman encoding) with encryption to reduce ciphertext size for bandwidth-critical applications.

These enhancements will position the system for real-time secure multimedia transmission in areas such as military communication, medical imaging, and AI dataset protection.

VIII. ACKNOWLEDGEMENT

Miss P. Bindhu Priya working as an Assistant professor in MCA in Sanketika Vidya Parishad Engineering College, Visakhapatnam, ap with 2.5 yrs teaching experience and member of IAENG, accredited by NAAC with her areas of interests in C, Data warehouse and data mining, Design and analysis of algorithm, python, Software Engineering

B. Narasimha Bhaskar is a final semester MCA student at Sanketika Vidya Parishad Engineering College, affiliated to Andhra University. With a strong interest in cryptography and Python-based application development, he carried out a project titled “An Enhanced ElGamal Cryptosystem for Image Encryption and Decryption.” His work focuses on secure image data transmission using public key cryptography, and he aspires to build a career in cybersecurity and software engineering.

REFERENCES:

[1] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” IEEE Transactions on Information Theory, vol. 31, no. 4, pp. 469–472, Jul. 1985.
[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. Pearson Education, 2017.
[3] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson Education, 2018.
[4] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. Wiley, 1996.
[5] H. Kaur and G. Singh, “Image encryption using ElGamal cryptosystem over finite field,” in *Proc. IEEE Int. Conf. on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, Mar. 2017, pp. 1761–1765.
[6] M. A. Khan and H. Farooq, “Enhanced ElGamal algorithm for secure image communication,” *International Journal of Computer Applications*, vol. 116, no. 6, pp. 1–6, Apr. 2015.
[7] P. Chouhan and R. Agrawal, “Performance evaluation of ElGamal encryption for image data,” *International Journal of Computer Applications*, vol. 162, no. 9, pp. 14–18, Mar. 2017.

[8] K. Patidar, M. Dave, and H. Agrawal, "A survey paper on image encryption techniques," *International Journal of Computer Applications*, vol. 107, no. 19, pp. 1–6, Dec. 2014.
[9] A. K. Jain, "Fundamentals of Digital Image Processing," Prentice-Hall, 1989.
[10] D. R. Stinson, *Cryptography: Theory and Practice*, 3rd ed. CRC Press, 2006.
[11] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
[12] N. Singla and G. Singh, "A new approach for image encryption using ElGamal algorithm," *International Journal of Engineering Trends and Technology*, vol. 11, no. 3, pp. 133–136, May 2014.
[13] S. S. Maniccam and N. G. Bourbakis, "Lossless image compression and encryption using SCAN," *Pattern Recognition*, vol. 34, no. 6, pp. 1229–1245, Jun. 2001.
[14] Python Software Foundation, "Python 3 Documentation," [Online]. Available: https://docs.python.org/3/
[15] OpenCV.org, "Open Source Computer Vision Library," [Online]. Available: https://opencv.org/
[16] NumPy Developers, "NumPy: Scientific computing tools for Python," [Online]. Available: https://numpy.org/
[17] F. Lundh, "Python Imaging Library (PIL)," [Online]. Available: https://pillow.readthedocs.io
[18] M. Kaur and S. Singh, "Study and analysis of image encryption techniques," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 5, pp. 786–790, May 2014.