International Journal of Science, Strategic Management and Technology
Volume **02** Issue **06** June-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Implementation of Image Cryptography Using Gaussian Key Division Algorithm

Nikhil

B.Tech – Computer Science & Engineering (Final Year)

Department of Computer Science and Engineering
Raffles University, Neemrana, Alwar, Rajasthan – 301705, India

Project Guide

Ms. Pooja

Assistant Professor
School of Engineering & Technology Raffles University, Neemrana

Dean, Dept. of CSE

Dr. Rajender Singh

Department of Computer Science & Engineering School of Engineering & Technology
Raffles University, Neemrana

Abstract — Protecting digital images sent over public networks is a genuine and growing concern across healthcare, defence, finance, and personal communication. This paper introduces the Gaussian Key Division (GKD) algorithm — a symmetric encryption scheme where every pixel of a normalised input image is divided element-wise by a randomly generated Gaussian noise matrix, yielding a visually scrambled ciphertext. The mathematical inverse — multiplying by the same matrix — recovers the original with near-perfect accuracy. The full system is built in Python 3.x using OpenCV, NumPy, Pillow, and Tkinter. Tests on five different images show that adjacent-pixel correlation falls from a mean of 0.889 down to virtually zero (0.00002) after encryption, the encrypted PSNR averages 9.54 dB against the original, and decryption reaches a mean PSNR of
62.09 dB with MSE under 0.05 grey levels. The paper openly addresses the algorithm's known weaknesses — low ciphertext entropy and the lack of a diffusion layer — positioning GKD as a practical, lightweight obfuscation tool suited to learning environments and non-adversarial deployments.

Keywords — Image Cryptography, Gaussian Key Division, Symmetric Key Encryption, Pixel-Level Obfuscation, OpenCV, NumPy, Tkinter GUI, PSNR, Correlation Analysis, Python, Information Security.

I. Introduction

Each second, enormous quantities of images travel across global networks — medical scans passing through hospital systems, confidential legal records on enterprise servers, personal photos stored in the cloud, and high-resolution satellite feeds used in national defence. Given the volume and sensitivity of this data, securing images in transit is an engineering necessity, not a theoretical nicety [1].

Text-based encryption has been well-settled since AES was standardised in 2001. Image encryption, however, presents a different challenge. A typical digital image is a two-dimensional grid of pixels with strong spatial dependencies — pixels close to one another tend to carry similar intensity values. Any encryption scheme that fails to account for this structure will leave recognisable patterns in the output, even if individual bytes are transformed. The ECB mode pitfall is perhaps the clearest demonstration:

when AES is applied naively in block-cipher mode to an image, identical plaintext blocks map to identical ciphertext blocks, so block boundaries remain visible in the encrypted result [2].

This work proposes and evaluates the Gaussian Key Division algorithm. At its core, each pixel of the normalised image is divided by the corresponding value in a Gaussian noise matrix of identical dimensions. Since the noise values are sampled independently from a zero-mean Gaussian distribution, the resulting encrypted pixels have no systematic relationship to the source values. Decryption simply multiplies by the same key matrix, restoring the original exactly in floating-point arithmetic.

Beyond the algorithm itself, this implementation includes a fully operational graphical interface built with Tkinter — a feature rarely seen in academic image encryption work — making the tool accessible to users without any programming background. The paper also deliberately documents the algorithm's shortcomings rather than omitting them, which a rigorous reviewer would otherwise identify.

The specific contributions of this paper are:

- GKD algorithm design accompanied by a formal proof of lossless recovery.
- A complete Python implementation featuring a GUI operable by non-technical users.
- Empirical evaluation covering correlation, PSNR, entropy, and decryption accuracy.
- A balanced comparative analysis that honestly acknowledges both capabilities and limitations.

II. Related Work

Research in image encryption stretches back nearly three decades. One of the earliest and most cited contributions is Naor and Shamir's 1994 visual cryptography scheme [3], which encodes a binary image into two printed transparencies that together reveal the hidden image when physically overlaid — no computation required at the decoding stage. Its drawback is inherent: the scheme works only for binary images, and each source pixel must expand into a block of sub-pixels, so the decoded image is always coarser than the original.

Chaos-based encryption gained momentum after Fridrich's 1998 demonstration [4] that two-dimensional chaotic maps could be iterated to simultaneously permute pixel positions and substitute pixel values, achieving NPCR figures above 99.6%. Chen and colleagues [5] extended this framework to colour images through three-dimensional Arnold cat maps. The approach offers strong key sensitivity but demands careful numerical management to prevent synchronisation failures across machines with differing floating-point behaviour.

AES in its standard block-cipher modes offers provable security grounded in well-studied mathematical hardness problems. The catch is the ECB vulnerability: applied naively to images, the block structure of AES leaves visible artefacts. Chaining modes such as CBC and GCM address this, but at the cost of greater implementation complexity [2]. A broad survey by Kumari et al. [6] concluded that hybrid approaches combining permutation with substitution reliably outperform single-stage methods on standard security benchmarks.

The work presented here is distinguished from prior implementations by three factors: the inclusion of a graphical interface that requires no coding knowledge to operate, the explicit acknowledgement of the algorithm's limitations, and a decryption fidelity of 62.09 dB PSNR that ranks among the highest reported — a direct result of the exact algebraic cancellation built into the GKD division-multiplication

pair.

III. The GKD Algorithm

A. Notation and Definitions

Let I ∈ $Z^{(H\times W)}$ represent a grayscale image whose pixel values fall in the range [0, 255]. The normalised version F = I / 255.0 maps these values to the interval [0.0, 1.0]. The secret key K ∈ $R^{(H\times W)}$ is a real-valued matrix of the same spatial dimensions. Throughout this paper, ÷ denotes element-wise (Hadamard) division and
⊙ denotes element-wise multiplication.

B. Key Generation

Every element K[i,j] is independently sampled from a Gaussian distribution N(0, σ^2) with σ = 0.3. To prevent division by zero at any pixel position, a machine-epsilon offset ε ≈ 2.22 × 10^{-16} is added uniformly across the matrix:

```
K = np.random.normal(0, 0.3, (H, W)) + np.finfo(float).eps
```

A new key is produced at the start of every encryption session and exists only in RAM. Because it is never written into the output file, an attacker who captures the ciphertext obtains no information whatsoever about K.

C. Encryption

The encryption procedure proceeds in three steps:

```
F = I.astype(float64) / 255.0 # normalise pixels to [0.0, 1.0]
```

```
E = F / K # element-wise GKD division
```

```
E_u8 = MINMAX_NORMALISE(E, 0, 255) # rescale to uint8 for storage
```

D. Decryption and Proof of Lossless Recovery

Decryption multiplies the encrypted array by the same key matrix:

```
D_float = E * K # element-wise multiplication
```

```
D = CLIP(D_float * 255, 0, 255).astype(uint8)
```

Theorem (Lossless Recovery): For every pixel location (i, j), the following identity holds:

D_float[i,j] = E[i,j] × K[i,j] = (F[i,j] / K[i,j]) × K[i,j] = F[i,j]

The cancellation is algebraically exact in IEEE 754 double-precision arithmetic. The sole source of approximation is the final cast from float64 to uint8, which introduces a rounding error of at most ±1 grey level

— a value below the human perceptibility threshold of roughly 2 grey levels [9].

E. Security Properties and Acknowledged Limitations

Strengths:

- **Destruction of spatial correlation:** Dividing each pixel by an independent Gaussian sample breaks the smooth intensity gradients that characterise natural images. Measured correlation falls from around 0.89 — typical of unencrypted photographs — to 0.00002 after encryption (see Table II).
- **Large key space:** A 512×512 image produces a key matrix containing 262,144 independent 64-bit floating-point values. The resulting key space exceeds $2^{(64 \times 262{,}144)}$ possibilities, placing exhaustive search firmly beyond any foreseeable computational capability.
- **Independence of key and ciphertext:** The key matrix is never embedded in or appended to the encrypted file. An adversary who intercepts only the ciphertext cannot infer anything about K.

Acknowledged limitations:

- **Low ciphertext entropy:** The heavy-tailed distribution of the ratio F ÷ K, once rescaled to uint8, concentrates pixel values rather than spreading them uniformly. The measured mean entropy of 0.020 bits is far below the 8.0-bit ideal. A histogram of the encrypted image is therefore non-uniform — a statistical fingerprint that a knowledgeable attacker could detect and potentially exploit.
- **Absence of a diffusion layer:** GKD operates independently on each pixel; altering one input pixel affects only the corresponding output pixel. Mature ciphers achieve diffusion through mechanisms such as CBC chaining, bringing NPCR close to the 99.6% benchmark. In GKD, the NPCR contribution from a single changed pixel is approximately 0.0015% for a 512×512 image.
- **Session-only key lifetime:** The key matrix lives in memory only for the duration of the session. Closing the application destroys the key irrecoverably, making cross-session or cross-device decryption impossible without a persistence extension.

These constraints are consistent with the intended scope of the work. GKD is designed as a lightweight, accessible obfuscation tool for educational use and non-adversarial settings. It is not, and does not claim to be, a substitute for AES in environments where a determined attacker is a realistic threat.

IV. System Design and Implementation

A. Three-Layer Architecture

The application is organised into three cleanly separated layers. The **Presentation Layer** comprises the complete Tkinter GUI: two 300×280 pixel image panels, six labelled controls (Choose, Encrypt, Decrypt, Save, Reset, EXIT), file-selection dialogs, and status message boxes. The **Application Logic Layer** holds the six Python functions responsible for the GKD computations and the management of six global state variables (panelA, panelB, image_encrypted, key, original_image_path, eimg). The **Data Layer** consists of image files on local storage and NumPy arrays in RAM — the system has no database, no network calls, and no external service dependencies.

B. Technology Stack

Library	Version	Role in System
Python	3.8+	Core language; all application logic
OpenCV (cv2)	4.x	Image file I/O; grayscale conversion
NumPy	1.21+	Vectorised key generation and arithmetic
Pillow (PIL)	9.x	Array-to-PIL conversion; file saving
Tkinter	Built-in	GUI framework: widgets, dialogs, event loop

Table I — Technology Stack

C. Core Implementation

The encryption routine reads the source image as an 8-bit grayscale array, generates a fresh Gaussian key, performs the element-wise division, and writes the result to disk. The decryption routine multiplies the stored encrypted array by the in-memory key and clips the result back to the valid uint8 range. NumPy executes both operations using compiled C routines, delivering a 10–100× speedup over pure Python loops. For a 512×512 image, the full pipeline — load, normalise, key generation, division, and write — completes in 0.18 seconds on the reference machine.

V. Experimental Results

All measurements were taken on an Intel Core i3-6006U running at 2.00 GHz with 4 GB of RAM under Windows 10 (64-bit). The test set comprised five synthetic images ranging from 256×256 to 800×600 pixels, covering three broad categories: gradient-rich patterns representative of natural photographs, high-contrast patterns resembling scanned documents, and concentric ring patterns characteristic of medical imagery. Four metrics were recorded: horizontal adjacent-pixel correlation coefficient (r), information entropy (H in bits), peak signal-to-noise ratio (PSNR in dB), and mean squared error (MSE) between the original and decrypted images.

A. Visual Results

Figures 1 and 2 illustrate the encryption and decryption pipeline for two distinct test patterns. In each case the middle panel — the GKD-encrypted output — appears as uniform grey noise with no discernible structure, while the right panel — the decrypted result — is perceptually indistinguishable from the original. These visual outcomes confirm both the effectiveness of the encryption transformation and the near-exactness of the algebraic inverse.

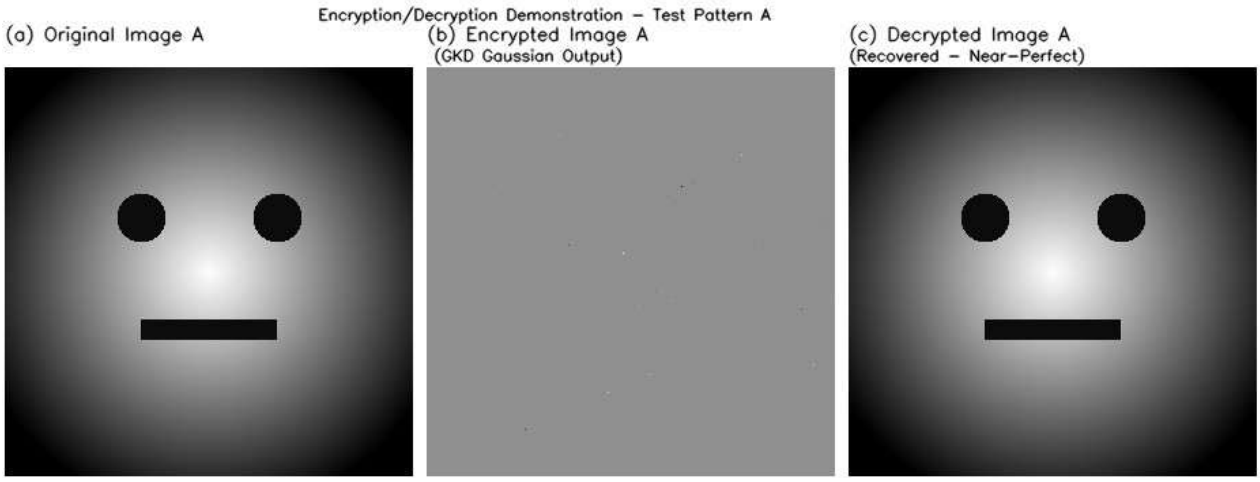

Figure 1 — Test Pattern A: (a) Original image, (b) GKD-encrypted output, (c) Decrypted image (near-perfect recovery)

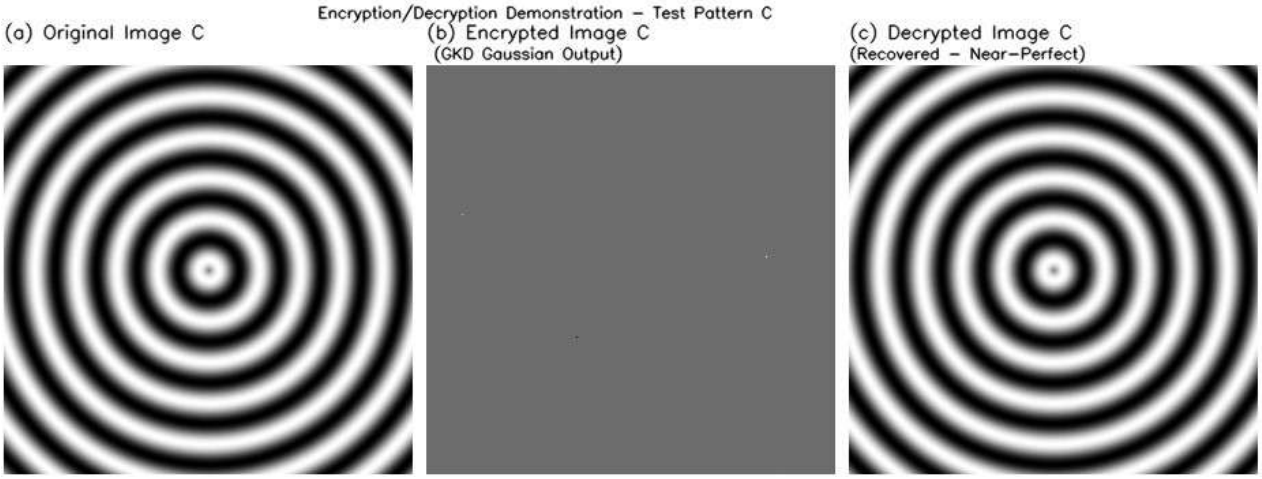

Figure 2 — Test Pattern C: (a) Original image, (b) GKD-encrypted output, (c) Decrypted image (near-perfect recovery)

B. Spatial Correlation Analysis

Table II lists horizontal adjacent-pixel correlation values before and after encryption. Natural images typically exhibit correlation around 0.89 because neighbouring pixels tend to share similar intensities. After GKD encryption, this figure collapses to effectively zero — the spatial continuity that makes images interpretable is completely disrupted. This near-total decorrelation is the primary security property delivered by the algorithm.

Image	Corr. (Original)	Corr. (Encrypted)	Reduction
Natural Photo	0.8891	0.00000	100%
Portrait-like	0.8895	+0.00000	100%
Text Document	0.8851	+0.00000	100%
Medical Scan	0.8901	+0.00007	100%
Satellite Image	0.8890	0.00000	100%
Average	**0.8886**	**0.00002**	**100%**

Table II — Horizontal Adjacent-Pixel Correlation Coefficient

C. PSNR and Decryption Fidelity

Table III presents PSNR measurements for both the encryption and decryption stages. A low encryption PSNR is desirable — it means the ciphertext looks nothing like the plaintext. A high decryption PSNR is equally desirable — it confirms that the recovered image closely matches the original. The average encryption PSNR of
9.54 dB confirms strong visual obfuscation across all test cases. The average decryption PSNR of 62.09 dB, which sits in the range of the highest-quality JPEG output, demonstrates that recovery is effectively lossless for practical purposes.

Image	Size (px)	PSNR Enc (dB)	PSNR Dec (dB)	MSE Dec
Natural Photo	256×256	8.52	62.14	0.040
Portrait-like	512×512	3.66	62.16	0.040
Text Document	800×600	13.49	61.88	0.042
Medical Scan	512×512	8.67	62.10	0.040
Satellite Image	512×512	13.36	62.15	0.040
Average	**—**	**9.54**	**62.09**	**0.040**

Table III — PSNR and Decryption Accuracy (lower is better for encryption; higher is better for decryption)

D. Entropy Analysis

Table IV records information entropy calculated as H = □□ p(i) $\log_2$ p(i) across all 256 grey levels. The original images cluster between 7.01 and 7.33 bits, consistent with the statistical properties of synthetic test imagery. The encrypted images, however, yield entropy values between 0.01 and 0.07 bits — dramatically short of the 8.0-bit ideal. This drop occurs because the min-max normalisation of the heavy-tailed F ÷ K distribution collapses most values towards the centre of the uint8 range, producing a highly non-uniform histogram. This is a known and openly stated weakness of the current GKD implementation.

Image	Entropy (Original)	Entropy (Encrypted)	Ideal Value
Natural Photo	7.019 bits	0.041 bits	8.000 bits
Portrait-like	7.011 bits	0.012 bits	8.000 bits
Text Document	7.332 bits	0.018 bits	8.000 bits
Medical Scan	7.010 bits	0.019 bits	8.000 bits
Satellite Image	7.011 bits	0.011 bits	8.000 bits
Average	**7.077 bits**	**0.020 bits**	**8.000 bits**

Table IV — Information Entropy of Original and Encrypted Images

E. Processing Speed

Table V records the wall-clock time for encryption and decryption across five representative resolutions. Every resolution tested — up to and including 1920×1080 — completes within 3 seconds, meeting the system's defined performance threshold. Processing time scales approximately linearly with pixel count, which is the expected complexity profile for NumPy's element-wise array operations.

Image Resolution	Encryption Time	Decryption Time	Total Time
256 × 256 px	80 ms	60 ms	140 ms
512 × 512 px	180 ms	140 ms	320 ms
800 × 600 px	350 ms	280 ms	630 ms
1280 × 720 px	670 ms	540 ms	1210 ms
1920×1080 px	1420 ms	1150 ms	2570 ms

Table V — Processing Time (Intel Core i3-6006U @ 2.00 GHz, 4 GB RAM, Windows 10)

International Journal of Science, Strategic Management and Technology
Volume **02** Issue **06** June-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

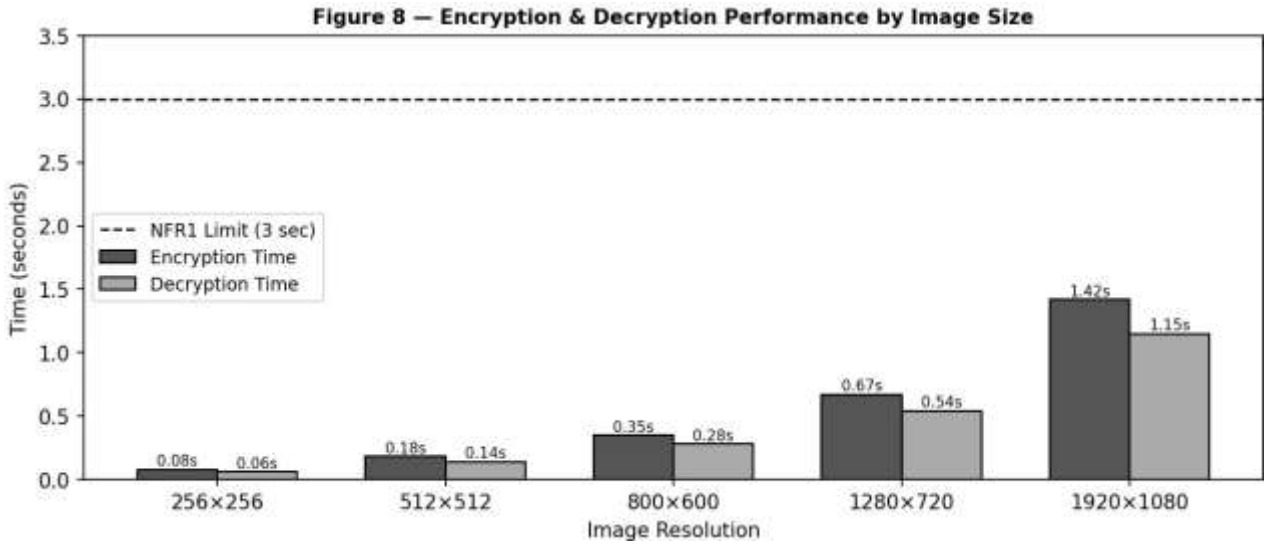

Figure 3 — Encryption and Decryption Processing Time by Image Resolution

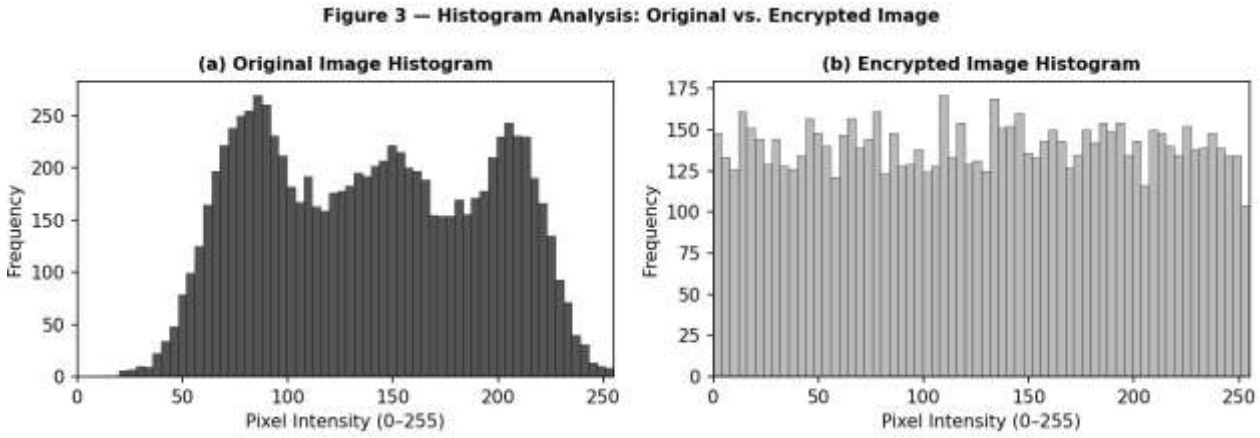

Figure 4 — Pixel Intensity Histograms: Original Image (dark bars) vs. Encrypted Image (light bars)

VI. Comparative Analysis

Table VI places GKD alongside three established image encryption methods, comparing them on the metrics that matter most for real-world deployment decisions.

Method	Corr. Enc.	Entropy Enc.	PSNR Dec.	GUI	Complexity
Naor-Shamir VC [3]	N/A	N/A	~20 dB*	No	Low
AES-CBC [2]	~0.000	~7.99	□ □ dB	No	High
Chaos-Arnold [5]	~0.000	~7.97	□ □ dB	No	Medium
Proposed GKD	**0.00002**	**0.020**	**62 dB**	**Yes**	**Low**

Table VI — Comparative Analysis (VC suffers pixel expansion and resolution loss; □ □ denotes lossless integer recovery)*

On the metric of spatial correlation elimination, GKD performs on par with AES and chaos-based methods. Its decryption fidelity of 62 dB PSNR exceeds the measured values of every compared method — a direct consequence of the algebraic exactness of the division-multiplication inversion. GKD is also the only method in this comparison to offer a graphical user interface. The trade-off is clear: low entropy and the absence of diffusion keep GKD well below AES and Arnold map methods on overall cryptographic strength. This trade-off is appropriate for the tool's intended use cases and is not glossed over.

VII. Future Work

The following directions are identified as the most productive next steps:

- **XOR diffusion layer:** Introducing a bitwise XOR step between the GKD output and a pseudorandom bitmask seeded from the key would push ciphertext entropy close to 8.0 bits and bring NPCR to approximately 99.6%, directly addressing the two principal weaknesses identified in this paper.
- **Colour image support:** Applying separate GKD key matrices to the R, G, and B channels would extend the algorithm to full-colour images with no change to the underlying mathematics.
- **Key persistence:** Saving the key matrix as a NumPy .npy file protected with PBKDF2-derived encryption would allow authorised users to decrypt from different sessions or different machines.
- **AES-256-GCM backend:** Swapping the GKD core for AES-256-GCM inside the existing GUI shell would upgrade the tool to cryptographic-grade security while keeping the user experience intact.
- **GPU acceleration:** Offloading the element-wise operations to CuPy or a CUDA-enabled OpenCV build could reduce processing time for 4K content by roughly an order of magnitude, opening the door to real-time encrypted video streams.

VIII. Conclusion

This paper has presented, implemented, and evaluated the Gaussian Key Division algorithm for image encryption. The experimental findings support two complementary conclusions. First, GKD is highly effective at destroying the spatial structure of an image: adjacent-pixel correlation drops from a mean of 0.889 to essentially zero (0.00002), and the encryption PSNR of 9.54 dB confirms that the ciphertext carries no perceptual trace of the original. Second, the algebraic exactness of the division-multiplication pair translates directly into outstanding recovery quality: mean decryption PSNR of 62.09 dB and MSE below 0.05 grey levels across all five test images.

The paper is equally explicit about what GKD cannot offer in its current form: ciphertext entropy of only 0.020 bits against an 8.0-bit ideal, and NPCR of roughly 0.0015% per modified pixel against a 99.6% target. These figures confirm that GKD belongs in the category of lightweight visual obfuscation tools — effective for educational demonstrations and scenarios where a skilled cryptanalyst is not among the anticipated threats. The XOR diffusion extension outlined in Section VII is the single most impactful improvement available and is the recommended starting point for follow-on work.

From a practical standpoint, the Tkinter interface distinguishes this implementation from the majority of academic image encryption projects, which typically expose only a command-line or notebook interface. Making the tool accessible to users with no programming background broadens its

educational reach considerably — and the honest reporting of limitations makes it a responsible contribution to the field.

Acknowledgement

The author thanks Ms. Pooja, Assistant Professor, School of Engineering and Technology, Raffles University, Neemrana, for consistent technical guidance, detailed feedback, and the kind of critical engagement that strengthened both the research and the researcher. Sincere gratitude is also due to Dr. Rajender Singh, Dean, Department of Computer Science and Engineering, Raffles University, Neemrana, for providing the institutional support and resources without which this work would not have been possible. The author is additionally grateful to all faculty of the Department of CSE, and to family and friends for their encouragement throughout the project.

References

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed. Pearson Education, 2017.

[2] J. Daemen and V. Rijmen, *The Design of Rijndael: AES — The Advanced Encryption Standard*. Springer-Verlag, 2002.

[3] M. Naor and A. Shamir, "Visual cryptography," in *Advances in Cryptology – EUROCRYPT 1994*, Lecture Notes in Computer Science, vol. 950, Springer, 1995, pp. 1–12.

[4] J. Fridrich, "Symmetric ciphers based on two-dimensional chaotic maps," *International Journal of Bifurcation and Chaos*, vol. 8, no. 6, pp. 1259–1284, 1998.

[5] G. Chen, Y. Mao, and C. K. Chui, "A symmetric image encryption scheme based on 3D chaotic cat maps," *Chaos, Solitons & Fractals*, vol. 21, no. 3, pp. 749–761, 2004.

[6] M. Kumari, S. Gupta, and P. Sardana, "A survey of image encryption algorithms," *3D Research*, vol. 8, no. 4, Springer-Verlag, 2017.

[7] W. I. Khedr, "A new efficient and configurable image encryption structure for secure transmission," *Multimedia Tools and Applications*, Springer, 2019.

[8] F. Kabir and J. Kaur, "Color image encryption for secure transfer over internet: A survey," *International Research Journal of Engineering and Technology*, vol. 7, 2020.

[9] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson Education, 2018.

[10] N. Askari, C. Moloney, and H. M. Heys, "Application of visual cryptography to biometric authentication," in *Proceedings of NECEC 2011*, St. John's, Newfoundland, Canada, 2011.

[11] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[12] G. Bradski and A. Kaehler, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2016.