JUIT ADMI Jurnal Ilmiah Teknik

JUIT Vol 5 No. 1 | January 2026 | ISSN: 2828-6936 (Print), ISSN: 2828-6901 (online), Page. 163-177

Chaos-Based Digital Image Encryption Using Arnold's Cat Map Permutation and Duffing Map Diffusion: A Python Desktop Implementation

Makmun[1*], Edi Sukirman[2], Erma Sova[3], Muhamad Wahyudi[3]
Technology Information, Gunadarma University, Depok, Indonesia

Article History
Received : January 04, 2026
Accepted : January 25, 2026
Published : January 26, 2026
Available Online:
January 26, 2026

Corresponding author*:
makmun@staff.gunadarma.ac.id

Cite This Article:
Makmun, M., Edi Sukirman, Erma Sova, & Muhamad Wahyudi. (2025). Chaos-Based Digital Image Encryption Using Arnold's Cat Map Permutation and Duffing Map Diffusion: A Python Desktop Implementation. Jurnal Ilmiah Teknik, 5(1), 163–177.

DOI:
https://doi.org/10.56127/juit.v5i1.2529

Abstract: Digital images are widely used for identification, communication, and information exchange, yet their large size, high pixel correlation, and uneven intensity distribution make them vulnerable to theft, manipulation, and statistical inference, thereby requiring encryption mechanisms tailored to image characteristics. **Purpose:** This study proposes a chaos-based permutation–diffusion scheme that combines Arnold's Cat Map for pixel permutation and the Duffing Map as a keystream generator for diffusion to strengthen digital image encryption and improve resistance to unauthorized analysis. **Methodology:** An experimental quantitative approach was conducted through the development of a Python-based desktop application. Ten test images (RGB and grayscale) with varying resolutions were encrypted and decrypted using predefined key settings. Security and performance were evaluated using histogram analysis, entropy, pixel correlation, key sensitivity, processing time, and key space. **Findings:** The encrypted images exhibit near-uniform histograms, entropy values approaching the ideal for 8-bit images (≈8), and pixel correlation values close to zero, indicating strong statistical concealment. The scheme also demonstrates high key sensitivity, where small key changes prevent meaningful decryption, and a large key space that supports brute-force resistance. Processing time increases with image size but remains practically feasible for desktop implementation. **Implications:** The proposed scheme can be applied to protect sensitive image data in local desktop environments and image exchange scenarios, reducing risks of statistical attacks and brute-force attempts while maintaining acceptable runtime for common image sizes. **Originality:** This study delivers an end-to-end integration of Arnold's Cat Map and the Duffing Map within a permutation–diffusion structure implemented as a Python desktop application, supported by structured security and efficiency evaluation, thereby providing a practical and reproducible contribution to chaos-based image cryptography.

Keywords: Image Encryption; Chaos-Based Cryptography; Arnold Cat Map; Duffing Map; Permutation–Diffusion; Python.

INTRODUCTION

Digital images are increasingly used as information media across personal and institutional activities, ranging from documentation and visual communication to various information-system services. As a consequence, digital images have become valuable data assets that are vulnerable to unauthorized access, manipulation, and misuse when stored or transmitted without adequate protection. Within the information-security perspective, this condition calls for mechanisms that can preserve confidentiality and prevent unauthorized parties from deriving meaning from visual content. Cryptography is commonly understood as an approach that transforms data into an encrypted form that is unreadable without a valid key (Stallings, 2017) and this principle is highly relevant to digital images as large-scale data objects (Munir, 2019), including image protection using cryptographic techniques.

Nevertheless, the characteristics of digital images mean that encryption cannot be treated in the same way as text encryption. Images typically involve large data volumes, high pixel correlation, and uneven intensity distributions, so an encryption scheme must suppress spatial structure and obscure statistical patterns while maintaining computational efficiency (Liu et al., 2019). Several image-encryption designs emphasize the need to randomize pixel positions and alter pixel values so that the encrypted output no longer reflects the visual or statistical traits of the original image (Zhang et al., 2019). In addition, nonlinear dynamical-system approaches have gained traction because they can produce seemingly random behavior and strong sensitivity to initial conditions properties that support key generation and strengthen encryption resistance (Pecora & Carroll, 2015; Strogatz, 2018). Consistently, chaos-based pseudo-random generator designs have been reported to produce sequences that are useful for diffusion processes in cryptography (Krishnamoorthi et al., 2021).

In terms of spatial scrambling, Arnold's Cat Map is often employed to transform pixel coordinates so that the positional regularity in an image becomes difficult to recognize, and it is commonly combined with other mechanisms to enhance overall encryption strength (Irawan & Rachmawanto, 2022; Rahmawati & Liantoni, 2018; Ratna et al., 2021) Meanwhile, strengthening the alteration of pixel intensity values can be achieved through diffusion using a keystream generated by a chaotic system, such as the Duffing Map, which has been used to support diffusion in image encryption (Zhou et al., 2014) and has demonstrated good performance in permutation–diffusion structures (Liu et al., 2019). To ensure practical usability and operability, an encryption scheme also needs to be realized as stable software, and Python provides an ecosystem that supports desktop application development along with user interfaces (Grayson, 2020; Sweigart, 2020).

Against this background, this study aims to design and implement a permutation–diffusion image-encryption scheme that combines Arnold's Cat Map at the permutation stage and the Duffing Map as a keystream generator at the diffusion stage. The scheme is implemented in a Python-based desktop application and evaluated using histogram, entropy, pixel correlation, key sensitivity, processing time, and key space as security and performance indicators (Liu et al., 2019; Zhang et al., 2019; Zhou et al., 2014).

The central argument of this study is that integrating Arnold's Cat Map and the Duffing Map can yield stronger image encryption: ACM-based permutation reduces spatial regularity and weakens pixel correlation, while Duffing-based keystream diffusion obscures intensity distributions and diminishes the feasibility of statistical analysis. Given the high parameter sensitivity of chaotic systems, the resulting key space is expected to be large and decryption infeasible without the correct key (Liu et al., 2019; Rahmawati & Liantoni, 2018; Zhou et al., 2014). Furthermore, implementing the scheme in a Python desktop application enables practical testing of both security and efficiency across different image sizes under realistic usage conditions (Grayson, 2020; Sweigart, 2020).

RESEARCH METHOD

This study adopts a digital image cryptography approach in which image data are treated as pixel-intensity matrices. Compared with text data, digital images typically have larger data volumes, strong pixel correlation, and non-uniform intensity distributions. Therefore, applying conventional block ciphers directly may not be computationally efficient or statistically optimal for image characteristics, motivating the use of permutation–diffusion structures that specifically target spatial correlation and intensity distribution leakage.

At the permutation stage, the proposed scheme employs Arnold's Cat Map (ACM) to shuffle pixel positions so that the spatial structure of the original image becomes unrecognizable. ACM is a two-dimensional chaotic map that is widely used for image encryption due to its simplicity and strong scrambling capability, while remaining reversible given the correct iteration setting. For an $N \times N$image, pixel coordinates $(x, y)$are mapped to new coordinates (x', y')using the linear modular transformation in Equation (1), where Ndenotes the image dimension and the number of iterations controls the scrambling strength.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} mod N$$

At the diffusion stage, the scheme uses the Duffing Map as a chaotic keystream generator. Duffing Map originates from the Duffing oscillator and exhibits strong chaotic behavior and high sensitivity to initial conditions, producing sequences that approximate random distributions and are suitable for cryptographic diffusion (Liu et al., 2019). The Duffing Map is defined recursively as shown in Equation (2), where $(x_0, y_0)$are initial conditions and $(a, b)$are control parameters. In this study, the Duffing system is operated under parameter settings that are known to induce chaotic behavior (e.g., $a = 2.75$ and $b = 0.2$), as illustrated by the bifurcation behavior in Figure 1 (Krishnamoorthi et al., 2021).

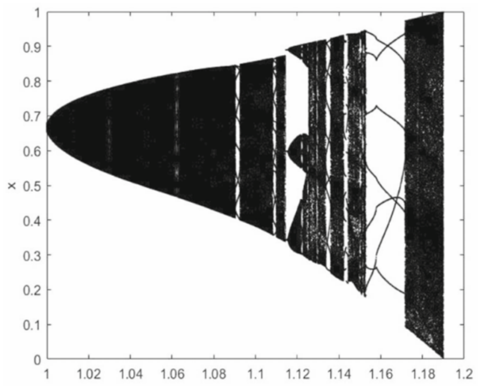


Figure 1. Bifurkasi Duffing Map

To apply the Duffing output to image diffusion, the generated real-valued sequence is normalized and quantized into the integer range 0–255 to form a keystream matrix $K(i, j)$aligned with image pixel positions. After ACM permutation, the diffusion process encrypts each permuted pixel value $P'(i, j)$by applying XOR with the corresponding keystream value, producing the ciphertext pixel $C(i, j)$as shown in Equation (3). Due to the involutive property of XOR, the same keystream enables correct decryption, while small key changes result in significantly different keystreams and prevent meaningful reconstruction (Liu et al., 2019).

RESULT AND DISCUSSION

Functional Validation of the Python Desktop Application and Encryption–Decryption Output

The proposed permutation–diffusion scheme was implemented as a Python-based desktop application to verify end-to-end operability and correctness. The application interface and workflow are presented in Figure 2 (cryptography application layout) and Algorithm 1 (Python implementation within the application).

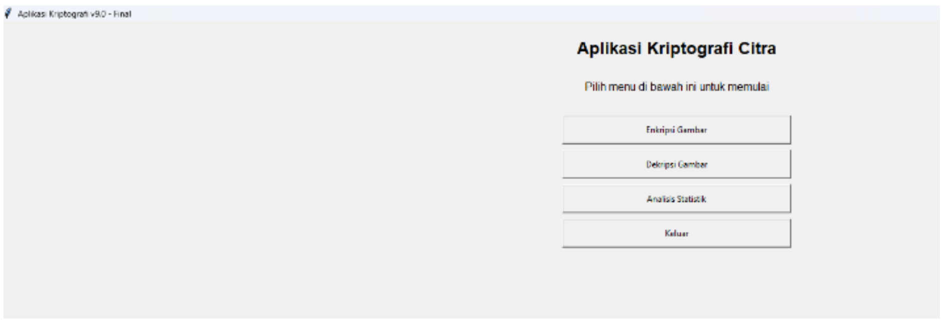

Aplikasi Kriptografi v9.0 - Final
Aplikasi Kriptografi Citra
Pilih menu di bawah ini untuk memulai
Enkripsi Gambar
Dekripsi Gambar
Analisis Statistik
Keluar

Figure 2. Image Cryptography Application Layout

Algorithm 1

```
class CryptoApp(tk.Tk):
    def __init__(self, *args,  kwargs):
        tk.Tk.__init__(self, *args,  kwargs)
        self.title("Aplikasi Kriptografi v9.0 - Final")
        self.geometry("1100x850")
        self.title_font = ("Arial", 18, "bold")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand=True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}
        for F in (HomePage, EncryptionPage, DecryptionPage, AnalysisPage):
            page_name = F.__name__
            frame = F(container, self)
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame("HomePage")
```

The encryption and decryption interfaces are shown in Figure 4 and Figure 5**.** Functional testing was conducted on both RGB and grayscale images to confirm that the system can transform readable images into visually unrecognizable ciphertext and then reconstruct the original images when the correct key is used. The main test configuration applied Duffing parameters $a = 2.75$, $b = 0.2$, and Arnold Cat Map iteration $t = 15$. The datasets used in this evaluation are summarized in Table 1 (RGB images) and Table 2 (grayscale images), while the qualitative outcomes and per-image processing time are reported in Table 3 (RGB) and Table 4 (grayscale).

Table 1. Color Image Test Data

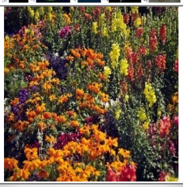
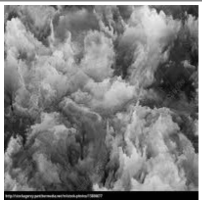
Data test No	Original Image	Image Name	Image Size (pixel)	Size File (*byte)*
1		Image1.jpg	199x199	82.2 KB
2		Image2.png	490x490	514.5 KB
3		Image3.png	502x502	456.98 KB
4		Image4.png	800x800	1.46 MB
5		Image5.bmp	1298x1298	4.82 MB

Table2. Greyscale Image Test Data

Data test No	Original Image	Image Name	Image Size (pixel)	Image Size (*byte)*
1		Image6.png	199x199	46.09 KB
2		Image7.png	490x490	514.5 KB

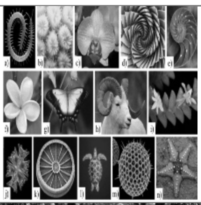
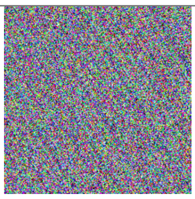
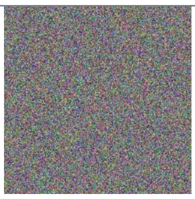
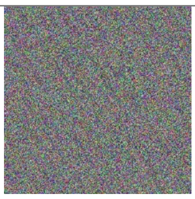
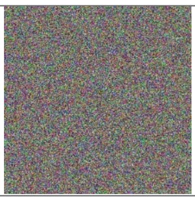
Data test No	Original Image	Image Name	Image Size (pixel)	Image Size (*byte)*
3		Image8.png	502x502	276.6 KB
4		Image9.png	800x800	974.06 KB
5		Image10.bmp	1289x1289	4.82 MB

Table 3. Results of Encryption and Decryption Testing of Color Images

Name	Original Image	Size	Encryption Result	Descryption Result
Image1.jpg		199x199	 Process time: 0.39 second	Process time: 0.42 second
Image2.png		490x490	 Process time: 2.60 second	Process time: 2.64 second
Image5.png		502x502	 Process time: 2.70 second	Process time: 2.82 second
Image4.png		800x800		

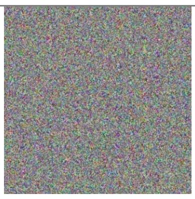
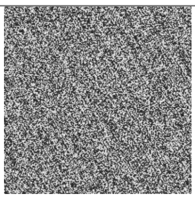
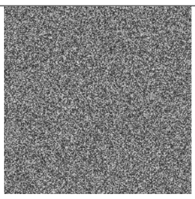
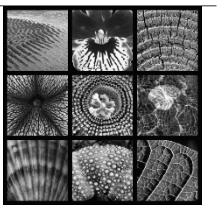
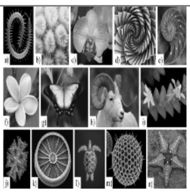
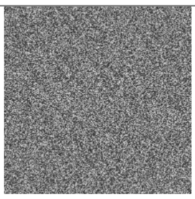
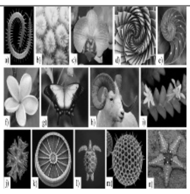
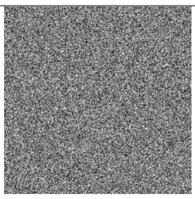
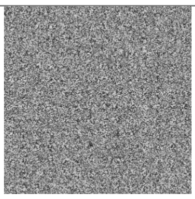
Name	Original Image	Size	Encryption Result	Descryption Result
			Process time: 7.17 second	Process time: 7.24 second
Image5.bmp		1298x1298	 Process time: 18.80 second	Process time: 18.98 second

Table 4. Results of Grayscale Image Encryption and Decryption Trials

Name	Original Image	Size	Encryption Result	Descryption Result
Image6.png		199x199	 Process time: 0.40 second	Process time: 0.43 second
Image7.png		490x490	 Process time: 2.56 second	 Process time: 1.63 second
Image8.png		502x502	 Process time: 2.72 second	 Process time: 2.82 second
Image9.png		800x800	 Process time: 5.24 second	Process time: 5.15 second

Name	Original Image	Size	Encryption Result	Descryption Result
Image10.bmp		1289x1289	 Process time: 18.17 second	Process time: 18.92 second

Across the experiments, ciphertext images consistently appeared noise-like and could not be interpreted visually, indicating that the encryption process effectively concealed the original spatial and intensity structures. When decryption was executed using the same key parameters, the original images were restored without observable loss, confirming reversibility under a correct key. Conversely, when the key was slightly altered, the decryption output turned into a random-looking image and failed to recover the original content, demonstrating strong key sensitivity. This behavior was observed consistently for both RGB and grayscale inputs, suggesting that the implementation is robust across different image modes.

Processing-Time Efficiency for RGB and Grayscale Images

Processing-time evaluation was performed to examine the practical efficiency of the implemented scheme under the same key configuration ($a = 2.75, b = 0.2, t = 15$) across varying image sizes. The average encryption and decryption runtimes for RGB images are reported in Table 5, and the corresponding results for grayscale images are reported in Table 6. The results show a clear relationship between resolution and runtime: small images (199×199 pixels) required approximately 0.39–0.43 seconds, medium images (around 490×490 to 800×800 pixels) required approximately 2.7–7.2 seconds, and large images (around 1290×1290 pixels) required approximately 18–19 seconds.

Table 5. Average Color Image Processing Time Test Results

Name Image	Size (piksel)	Average Encryption Processing Time (seconds)	Average Decryption Time (seconds)
Image1.jpg	199x199	0.39 second	0.42 second
Image2.png	490x490	2.60 second	2.64 second
Image3.png	502x502	2.70 second	2.82 second
Image4.png	800x800	7.17 second	7.24 second
Image5.bmp	1298x1298	18.80 second	18.98 second

Table 6. Average Grayscale Image Processing Time Test Results

Name Image	Size (piksel)	Average Encryption Processing Time (seconds)	Average Decryption Time (seconds)
Image6.png	199x199	0.40 second	0.43 second
Image7.png	490x490	2.56 second	1.63 second
Image8.png	502x502	2.72 second	2.82 second
Image9.png	800x800	5.24 second	5.15 second
Image10.bmp	1289x1289	18.17 second	18.92 second

Overall, encryption and decryption times were relatively balanced, indicating comparable computational complexity in both directions for the same key. Minor runtime deviations in some grayscale cases may be influenced by implementation overhead, file I/O, or system-level factors rather than algorithmic differences. These findings indicate that the approach remains practical for small-to-medium images in desktop usage, while higher-resolution inputs would benefit from computational optimization in future work.

Statistical Security Evidence Based on Histogram Analysis

To assess whether encryption obscures intensity distribution patterns that could be exploited in statistical attacks, histogram analysis was performed by comparing the original and encrypted images. The histogram of an original image is shown in Figure 6**,** while the histogram of its encrypted counterpart is shown in Figure 7**.** The original histogram exhibits a non-uniform distribution that reflects the visual characteristics of the plaintext image, whereas the encrypted histogram becomes markedly more uniform and random-like, indicating that the encryption process reduces statistical leakage from intensity distributions.

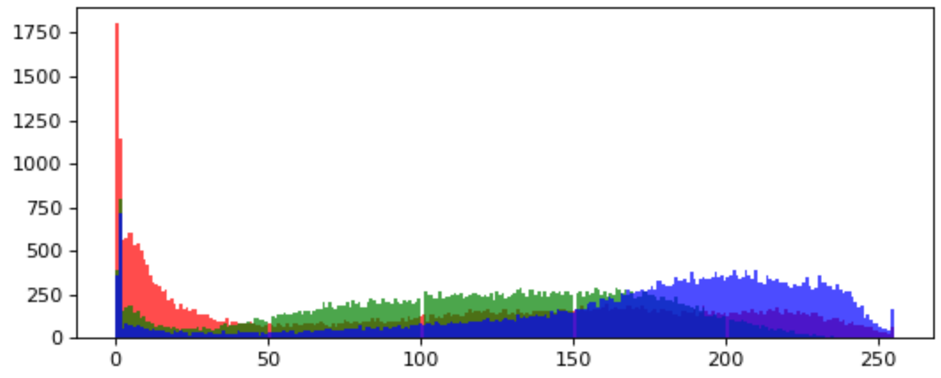


Figure 6. Histogram of Original Image Image1.Png

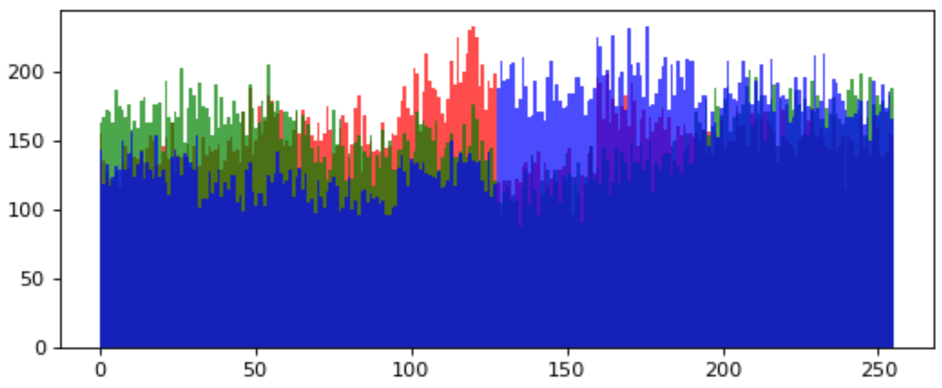


Figure 7. Encrypted Image Image1.Png

This tendency supports the intended role of the permutation–diffusion structure: the permutation disrupts spatial regularity, while diffusion alters pixel intensities so that ciphertext no longer retains easily exploitable distributional signatures. The observed histogram flattening aligns with expected behavior in chaos-based image encryption and indicates improved resistance to histogram-based statistical analysis (Liu et al., 2019; Ratna et al., 2021).

Discussion

This study examined a chaos-based permutation–diffusion image-encryption scheme that combines Arnold's Cat Map for pixel permutation and the Duffing Map for keystream-driven diffusion, implemented in a Python desktop application. The results confirm three central outcomes. First, the application reliably transforms plaintext images into ciphertext images that are visually unrecognizable and restores the original images when decryption uses the correct key parameters. Second, processing time increases with image size, indicating that the approach is practical for small-to-medium resolutions but becomes more time-consuming for high-resolution inputs. Third, histogram evidence shows that ciphertext distributions become more uniform, suggesting reduced statistical leakage and improved resistance to histogram-based attacks (Liu et al., 2019; Ratna et al., 2021).

These outcomes occur due to the complementary roles of permutation and diffusion in the proposed structure. Arnold's Cat Map operates on pixel coordinates, disrupting spatial regularities that naturally exist in images due to neighborhood similarity and high pixel correlation. By repeatedly scrambling pixel positions through controlled iterations, ACM reduces the visibility of the original structure even before any change to intensity values. The Duffing Map then strengthens security through diffusion by producing a keystream that is highly sensitive to key parameters and initial conditions, which after normalization modifies pixel intensities via XOR so that the ciphertext no longer preserves the original intensity distribution. This combination explains why ciphertext appears noise-like, why minor key changes cause decryption failure, and why the encrypted histogram becomes flatter: spatial structure is dismantled by permutation, while intensity-based statistical signatures are masked by diffusion (Liu et al., 2019).

In comparison with prior studies, the findings align with established evidence that permutation–diffusion schemes improve image-encryption security by simultaneously suppressing spatial correlation and intensity statistics. For example, Liu, Wang, and Kadir report that Duffing-map-based permutation–diffusion designs can produce near-ideal statistical indicators (e.g., increased entropy and reduced correlation), and Zhou, Panetta, Agaian, and Chen demonstrate that Duffing-chaos components can be integrated with other structures to strengthen encryption against attacks (Zhou et al., 2014). Likewise, research combining Arnold's Cat Map with other chaotic maps for diffusion has shown that ACM is effective for confusion but benefits from being paired with diffusion to mitigate statistical analysis (Irawan & Rachmawanto, 2022; Rahmawati & Liantoni, 2018; Ratna et al., 2021). The novelty of the present study lies in (i) integrating ACM (permutation) and Duffing Map (keystream diffusion) into a single end-to-end scheme, (ii) operationalizing the design as a Python desktop application to demonstrate practical usability (Grayson, 2020; Sweigart, 2020) and (iii) reporting functional validation, runtime behavior across multiple image sizes, and histogram-based statistical evidence within the same experimental workflow, thus strengthening the applied contribution beyond purely theoretical descriptions.

From a broader interpretive standpoint, the results reinforce the view that safeguarding visual data requires encryption mechanisms tailored to the statistical nature of images rather than relying solely on conventional assumptions derived from text security. In social and institutional contexts where images are increasingly used for verification, documentation, and communication, encryption that effectively removes spatial and statistical signatures can reduce the risk of privacy harm and unauthorized inference. Technically, the results also contribute to a wider understanding of how chaos-based dynamics through sensitivity and nonlinear behavior can be harnessed to produce strong keystreams suitable for diffusion-driven image (Krishnamoorthi et al., 2021; Pecora & Carroll, 2015; Strogatz, 2018).

At the same time, the study also reveals practical trade-offs. The functional benefit of producing visually secure ciphertext and demonstrating high key sensitivity is accompanied by increasing computational cost at larger resolutions. While the approach remains efficient for smaller images, runtime grows substantially for large images, which may limit real-time deployment or batch processing in high-throughput environments. In addition, chaos-based encryption imposes operational requirements for secure key management: parameter leakage or weak key-setting practices can undermine security even if the algorithm is statistically strong. Thus, the method functions well as a protection mechanism, but its effectiveness depends on how it is operationalized in realistic usage scenarios.

Based on these findings, several action-oriented implications can be proposed. First, for practical deployment, the system should incorporate key-management guidelines and enforce minimum key-quality constraints (e.g., non-trivial iteration counts and parameter precision) to avoid predictable settings (Munir, 2019; Stallings, 2017). Second, performance optimization should be prioritized for high-resolution images, for example by improving pixel-level operations through vectorization and more efficient memory handling in Python-based implementations (Grayson, 2020; Sweigart, 2020). Third, evaluation should be expanded beyond histogram evidence to include additional standard security indicators (e.g., entropy and pixel-correlation tables across directions, as commonly used in chaos-based encryption studies) to provide stronger evidence for reviewers and facilitate comparison across methods (Liu et al., 2019; Ratna et al., 2021). Collectively, these steps would improve both the scientific rigor and practical readiness of the proposed approach for secure image handling in real-world applications.

CONCLUSION

This study demonstrates that combining Arnold's Cat Map (permutation) and the Duffing Map (keystream-driven diffusion) can produce a practical and secure approach for digital image encryption and decryption within a desktop environment. The main takeaway is that the proposed scheme successfully transforms plaintext images into visually unrecognizable ciphertext and reliably reconstructs the original images when the correct key is applied. From a performance perspective, processing time increases with image size: for small images (199×199 pixels), encryption requires approximately 0.39–0.40 seconds and decryption 0.42–0.43 seconds; for medium images (502×502 to 800×800 pixels), the runtime increases to approximately 2.7–5.2 seconds; and for large images (around 1290×1290 pixels), encryption and decryption take about 18–19 seconds. Overall, the results indicate that the ACM–Duffing integration supports strong confidentiality properties and is designed to resist brute-force and statistical attacks.

The primary scientific contribution of this work lies in delivering an end-to-end permutation–diffusion scheme that integrates ACM and Duffing Map into a single pipeline and operationalizes it as a Python-based desktop application. Beyond conceptual design, the study provides applied evidence through functional validation (encryption–decryption correctness), runtime behavior across multiple image sizes, and histogram-based statistical security characteristics, thereby strengthening the practical relevance of chaos-based image cryptography for real-world use.

This study also has limitations. The statistical security evaluation presented is still partial, as it emphasizes histogram evidence while additional quantitative indicators commonly used in image cryptography such as entropy values and directional pixel-correlation metrics for each test image are not fully reported in tabular form. Moreover, the current implementation experiences a substantial runtime increase for high-resolution images, indicating a need for optimization if real-time or large-scale processing is required. Future research should therefore extend the evaluation with comprehensive entropy and correlation reporting, include additional robustness tests (e.g., plaintext sensitivity metrics), and improve computational efficiency through algorithmic and implementation-level optimization to support broader deployment scenarios.

REFERENCES

Grayson, J. (2020). *Python and Tkinter Programming*. Manning Publications.

Irawan, C., & Rachmawanto, E. H. (2022). Implementasi Kriptografi dengan Menggunakan Algoritma Arnold's Cat Map dan Henon Map. *Jurnal Masyarakat Informatika*, *13*(1), 15–32. https://doi.org/10.14710/jmasif.13.1.43312

Krishnamoorthi, S. et al. (2021). Design of pseudo-random number generator from turbulence padded chaotic map. *Nonlinear Dynamics*, *104*(2), 1627–1643. https://doi.org/10.1007/s11071-021-06346-x

Liu, H. et al. (2019). A novel image encryption scheme based on Duffing map and permutation–diffusion structure. *Journal of Ambient Intelligence and Humanized*

Computing, *10*(12), 4811–4825.

Munir, R. (2019). *Kriptografi*. Informatika Bandung.

Pecora, L. M., & Carroll, T. L. (2015). Synchronization of chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, *25*(9), 97611. https://doi.org/10.1063/1.4917383

Rahmawati, W. M., & Liantoni, F. (2018). Penggunaan Arnold Cat Map Dan Beta Chaotic Map Pada Enkripsi Data Citra. *Jurnal ELTIKOM : Jurnal Teknik Elektro, Teknologi Informasi Dan Komputer*, *2*(2), 50–57. https://doi.org/10.31961/eltikom.v2i2.85

Ratna, A. A. P. et al. (2021). Chaos-Based Image Encryption Using Arnold's Cat Map Confusion and Henon Map Diffusion. *Advances in Science, Technology and Engineering Systems Journal*, *6*(1), 316–326. https://doi.org/10.25046/aj060136

Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson.

Strogatz, S. H. (2018). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. CRC Press.

Sweigart, A. (2020). *Automate the Boring Stuff with Python*. No Starch Press.

Zhang, Y. et al. (2019). An image encryption scheme based on rotation matrix bit-level permutation and block diffusion. *The Journal of Supercomputing*, *75*(10), 6293–6311.

Zhou, Y. et al. (2014). Image encryption algorithm based on Duffing chaotic map and DNA sequence. *Optik*, *125*(18), 5455–5460.