# Adaptive Encryption and Decryption Framework for Multi-Modal Data Using AES-CBC and ChaCha20

Harshvardhan Handa[1], Deepika Kukreja[2], Suryansh Choudhary[3], Amisha Kapoor[4]
*Department of Information Technology Netaji Subhas University of Technology* Delhi, India

***Abstract*—The explosion of diverse multimedia content de- mands encryption solutions that can adapt to different data formats while maintaining security and performance. This re- search presents an adaptive encryption system combining a desktop GUI (Tkinter) and a web-based platform (Flask) that intelligently selects encryption algorithms based on file type: AES-CBC for structured data (text, images) and ChaCha20 for streaming media (audio, video). The system features dynamic key derivation using PBKDF2 with unique salts, user authentication, secure session handling, and metadata tracking. Experimental evaluations demonstrate that this architecture provides strong security, efficient performance, and flexibility for securing het- erogeneous files across local and web-based environments.**

***Index Terms*—adaptive encryption, AES-CBC, ChaCha20, multimedia security, Flask, Tkinter, PBKDF2, file encryption**

## I. INTRODUCTION

The growth of multimedia data across personal, cloud, and enterprise environments necessitates encryption solutions that can adapt to varying file types without compromising efficiency. Traditional encryption strategies applying uniform ciphers across all data often introduce unnecessary perfor- mance bottlenecks or weaken security guarantees. Structured, static data such as text documents and images benefit from block cipher modes like AES-CBC [1], while continuous, unstructured streams like audio and video are more effectively secured with stream ciphers like ChaCha20 [2].

Addressing this need, we propose an adaptive encryption framework featuring two integrated systems: a local GUI- based desktop application built with Tkinter and a scalable web platform developed in Flask. Both systems perform intelligent algorithm selection based on file analysis, ensuring optimal security-performance tradeoffs. Key management is enhanced through PBKDF2-based dynamic key derivation with unique per-user salts, while session control, login au- thentication, and metadata @recording ensure a complete end-to- end secure solution. Our system provides a flexible encryption mechanism suitable for personal, organizational, and cloud- driven use cases.

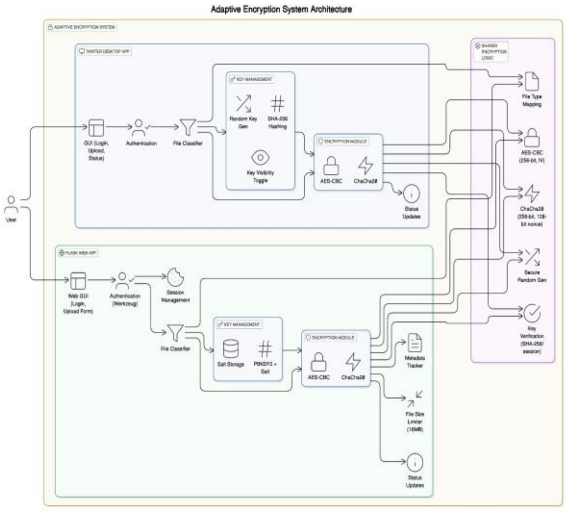

Fig. 1. Adaptive-Architecture

Multimedia encryption methods have evolved from tradi- tional full encryption strategies to more targeted, context- aware mechanisms. Conventional models using symmetric encryption (e.g., AES-ECB) for entire data streams suffer from visual pattern leakage and performance degradation, especially in mobile or resource-constrained environments [3]. Research into selective encryption approaches, where critical data segments are encrypted selectively, introduced efficiency but often at the cost of reduced security.

Block ciphers like AES-CBC remain the gold standard for static data encryption, with numerous studies confirming their reliability when combined with secure padding schemes. Stream ciphers like ChaCha20, standardized under RFC 7539, have shown significant advantages in low-latency applications, offering robust resistance against side-channel attacks while maintaining high throughput. Comparative studies highlight ChaCha20's performance superiority in environments lacking hardware-accelerated AES capabilities.

Recent work also emphasizes adaptive security models, where encryption strength dynamically adjusts based on file type, threat level, or user profile. The use of PBKDF2 and Scrypt-based key derivation functions in adaptive frameworks enhances resilience against password-based attacks by increas- ing computational costs [4]. Flask and Tkinter have emerged as popular frameworks for deploying secure encryption work- flows in web-based and local contexts respectively, owing to their flexibility and community support.

Our work integrates these advancements into a practical, user-friendly encryption system that adapts algorithm selection and key management strategies based on data properties and user input, while maintaining lightweight operational over- head.

## II. LITERATURE REVIEW

With the rapid growth in the generation and transmis- sion of multi-modal data—such as text, images, audio, and video—there is a growing need for adaptive encryption frame- works that tailor cryptographic algorithms to data-specific characteristics. AES-CBC (Advanced Encryption Standard in Cipher Block Chaining mode) and ChaCha20 have emerged as prominent choices for encrypting structured and streaming data, respectively.

A comparative study by Muhammed et al. [5] evaluated the performance of AES, Blowfish, Twofish, Salsa20, and ChaCha20 in image encryption. Their results demonstrated that ChaCha20 outperformed the other ciphers in terms of speed, especially for large images. However, the study focused primarily on timing and lacked a deep analysis of security robustness.

In a separate work, Elbadawi and Hadi [6] optimized AES- CBC for wireless sensor networks, achieving up to a 30% increase in speed through software enhancements. However, this optimization came at the cost of doubled memory usage, which could be a constraint in low-memory environments.

Modifications to the AES algorithm have been explored to support high-definition image encryption more efficiently. Sharma and Sabharwal [7] proposed a tailored AES scheme for large images, focusing on reducing hardware require- ments. Yet, the study did not provide detailed comparative benchmarks or analyze the impact on real-time performance. Similarly, an image encryption method integrating AES with a chaotic system was proposed by Rani and Saravanan [8], which improved resistance against differential attacks and demonstrated better NPCR and UACI metrics. Nonetheless, the added complexity from chaos system integration intro- duced overhead and potential scalability challenges.

For streaming data like video, ChaCha20 has shown promis- ing results. Faragallah et al. [9] utilized ChaCha20 in a selec- tive encryption framework for HEVC video streams, aimed at secure telehealth applications. Their method preserved visual fidelity and achieved low latency, making it suitable for real-time communication. However, its application was domain-
specific, and scalability across broader systems was not ad- dressed.

Another study by Al-Khafaji and Hussein [10] compared block cipher modes such as OCB combined with AES and Serpent on digital images. While offering strong performance, the analysis lacked in-depth security evaluations of OCB's vulnerabilities in diverse data environments.

Lastly, weak points in earlier chaos-based encryption sys- tems were critically assessed by Torkaman and Eslami [11]. They identified that many such systems required multiple iterations to ensure proper avalanche effect and recommended block-level improvements. However, their proposed fixes in- creased computational overhead and may not be practical for lightweight or real-time use cases.

In summary, the reviewed studies underscore the comple- mentary roles of AES-CBC and ChaCha20 in an adaptive encryption framework for multi-modal data. While AES-CBC remains effective for structured data types like text and images, ChaCha20's efficiency makes it a strong candidate for real- time audio and video encryption. The literature also highlights common limitations such as memory usage, computational complexity, and limited real-world scalability—factors that must be carefully considered in implementing a unified, adaptive encryption system.

## III. METHODOLOGY

### *A. System Architecture and Implementation*

The adaptive encryption system comprises two components:

1) Tkinter-Based Desktop Application:

This standalone GUI application provides local file encryption and decryption services. It includes login authentication, secure random key generation, and dynamic key hashing using SHA-256 [12]. Users upload files, which are classified based on extensions to determine the appropriate encryption algorithm: AES- CBC for text and images, ChaCha20 for audio and video. The application supports key visibility toggling and provides detailed status updates throughout the encryption process.

2) Flask-Based Web Application:

The web server application offers remote file encryption via browser interface. User accounts are secured with hashed passwords using Werkzeug's password hashing utilities, while encryption keys are dynamically derived using PBKDF2 with unique salts per user. Files are uploaded through web forms, encrypted appropriately based on their MIME type or extension, and returned to the user. Metadata including encryption type, salt usage, and encryption timestamp is tracked internally.

Both systems use AES with 256-bit keys in CBC mode and ChaCha20 with 256-bit keys and 128-bit nonces. Secure ran- dom IVs and nonces are generated per encryption operation.

The web platform also limits maximum upload file size to 16 MB to ensure performance consistency.

### *B. Adaptive Encryption and Decryption Logic*

Upon file upload, the system classifies the file:

- Text, PDF, DOC, DOCX → AES-CBC
- PNG, JPG, JPEG, GIF → AES-CBC
- MP3, WAV, OGG, MP4, AVI, MKV → ChaCha20

During decryption, users must specify the file type, enabling the system to apply the correct decryption logic. Both applica- tions verify encryption key correctness using SHA-256 based checks or explicit session matching mechanisms.

### *C. Key Management*

Key generation follows a secure derivation process:

- Desktop App: Manual random key generation + optional password processing through SHA-256.
- Web App: Password input processed through PBKDF2HMAC (100,000 iterations, SHA-256) using user-specific salts stored during registration.

Keys are never stored in plaintext and are dynamically derived per session or operation.

### *D. User Authentication and Session Management*

The web system manages user sessions through Flask's built-in session cookies, secured with secret keys. Passwords are never stored directly; instead, password hashes generated via Werkzeug are retained, ensuring strong password security.

## IV. IMPLEMENTATION

The adaptive encryption framework was developed in two complementary modules to support both local and remote use cases: (1) a standalone Tkinter-based Desktop Application and

(2) a Flask-based Web Application. Both modules were imple- mented using Python 3.11 and rely on secure cryptographic libraries, with emphasis on modularity, usability, and strong security guarantees.

### *A. System Overview*

Both applications follow a layered architectural design, consisting of:

- User Interface Layer: Implemented using Tkinter (for the desktop app) or HTML templates served by Flask (for the web app).
- Controller Layer: Handles user authentication, file up- loads, encryption mode selection, and general workflow management.
- Cryptographic Engine Layer: Executes core operations including key derivation, encryption, decryption, and secure random number generation.

This layered approach supports maintainability, extensibil- ity, and unit test coverage for critical subsystems.

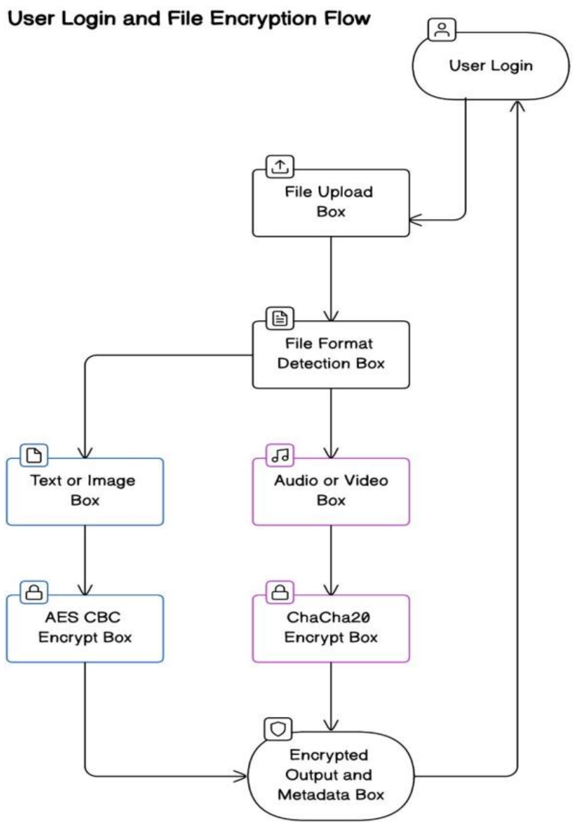

Fig. 2. Workflow

*B. Encryption and Decryption Engine*

The encryption engine supports two symmetric algorithms:

- AES in CBC Mode: Applied to text and image files. Utilizes a 256-bit key and a 128-bit Initialization Vector

(IV) generated per session.

- ChaCha20 Stream Cipher: Applied to audio and video files. Utilizes a 256-bit key with a 128-bit random nonce for each operation.

*1) AES-CBC Encryption Workflow:*

1) Input data is padded using PKCS7 to align with AES's 128-bit block size.
2) A 16-byte IV is generated using os.urandom().
3) Encryption is performed using the Cipher class from the cryptography.hazmat module.
4) The IV is prepended to the ciphertext for use during decryption.

*2) ChaCha20 Encryption Workflow:*

1) A 16-byte nonce is randomly generated per operation.
2) Data is encrypted using the ChaCha20 cipher (no padding required).
3) The nonce is prepended to the ciphertext to allow correct decryption.

Decryption for both algorithms involves extraction of the IV/nonce and reinitialization of the cipher context before decrypting the ciphertext.

*C. Key Derivation and Management*

Key derivation ensures that user passwords are never used directly as encryption keys. Instead, secure key derivation functions (KDFs) are employed:

- PBKDF2-HMAC-SHA256: Used across both applica- tions, configured with 100,000 iterations.
- Salt Management: A unique 16-byte salt is generated for each user (Flask app) or each session (Tkinter app).

The derived key is 32 bytes (256 bits) in length and is used for both AES and ChaCha20 operations. The Flask app stores only password hashes (via Werkzeug) and salts—raw keys are never persisted.

*D. Desktop Application (Tkinter GUI)*

The desktop application provides a native, offline encryption environment.

- Login Module: Includes a basic admin authentication interface to guard access.
- Key Management Panel: Allows users to manually input or generate a random 256-bit key. Toggle options allow key visibility for usability.
- File Processing Module: Uses a file dialog for file selection. Based on file extension, the correct encryption algorithm (AES-CBC or ChaCha20) is applied. Feedback is provided via a scrollable text widget.
- Error Handling: Built-in validation detects incorrect keys, unsupported formats, and corrupt files, displaying descriptive popup messages.

This module is tailored for offline encryption needs with no dependency on internet connectivity.

*E. Web Application (Flask Server)*

The Flask application delivers encryption services over the web with built-in session and user management.

- Authentication System: Users register with a username and password. Passwords are securely hashed and stored; sessions are managed via Flask cookies.
- Encryption Workflow: Files are uploaded through HTML forms. Based on MIME type and extension,

either AES-CBC or ChaCha20 is applied. Encrypted files are returned in base64-encoded form.

Decryption Workflow: Users upload encrypted files and specify the original file type and password. The system derives the decryption key and reverses the process.

- Metadata Tracking: Internal logs capture salt, key derivation method, encryption timestamp, and file type

for audit and debugging.

- Security Measures: Upload file size is limited to 16 MB. Application-wide secret keys protect session cookies. Flash messages ensure secure user feedback.

This module is suited for users requiring encryption without installing additional software.

### *F. Cryptographic Libraries Used*

- cryptography: The primary backend for encryption (AES, ChaCha20), key derivation (PBKDF2HMAC), and padding operations.
- PyCryptodome: Used in the desktop application for simplified symmetric encryption operations.
- Werkzeug: Used in the web application for secure pass- word hashing and verification.

All cryptographic operations utilize secure random number generation (os.urandom) and follow industry best practices (e.g., NIST and IETF guidelines).

## V. RESULTS

The adaptive encryption framework was evaluated for its encryption speed, computational resource usage, encryption overhead, and security levels across different data types. A comparative analysis was also conducted against other widely used encryption algorithms including AES-GCM, Blowfish, and Triple DES to establish the effectiveness of AES-CBC and ChaCha20 within the adaptive system.

The primary performance evaluation involved encrypting

40 sample files, categorized into text documents, images, audio, and video files, ranging from 100KB to 16MB in size. Encryption and decryption speeds were measured, CPU and memory usage were profiled, and file size impacts due to encryption overhead were analyzed.

Encryption speed tests revealed that ChaCha20 consistently outperformed AES-CBC on larger audio and video files, achieving an average throughput of 110–120 MB/s. AES- CBC showed reliable performance on smaller, static files like text and images, averaging 75–85 MB/s. Resource profiling indicated that ChaCha20 consumed slightly less CPU (average 25–30% utilization) compared to AES-CBC (average 35–40% utilization) during encryption of large media files. Memory usage remained moderate across all tests, peaking at around 300MB.

To evaluate the performance of the proposed adaptive en- cryption system, a comprehensive experimental analysis was conducted using a standardized corpus of text, image, audio, and video files. Each encryption algorithm was assessed across three primary metrics: encryption time ($T_{\text{enc}}$), average CPU utilization ($\text{CPU}_{\text{avg}}$), and encrypted output file size. These metrics were selected to reflect real-world trade-offs between speed, resource efficiency, and ciphertext overhead.

To enable a unified comparison, we introduce a composite performance metric: encryption efficiency $\eta$, defined as:

$$\eta = \left( \frac{\vec{S}}{T_{\text{enc}}} \right) \times \left( 1 - \frac{\Delta S}{S} \right) \times \left( 1 - \frac{\text{CPU}_{\text{avg}}}{100} \right) \times 100 \rightarrow (1)$$

where:

- $S$ is the original file size (in bytes),
- $T_{\text{enc}}$ is the encryption time (in seconds),
- $\Delta S$ is the increase in size after encryption,
- $\text{CPU}_{\text{avg}}$ is the average CPU utilization during encryption (in percentage).

This metric penalizes excessive CPU usage and ciphertext expansion, while rewarding faster processing speeds. $\eta$ is ex- pressed as a percentage, with higher values indicating superior overall encryption efficiency.

### *A. Static File Types: Text and Image*

Table I summarizes the performance of five symmetric encryption algorithms — AES-CBC, AES-GCM, ChaCha20, Blowfish, and Triple DES — for static file types where structured data and block alignment are critical.

TABLE I
ENCRYPTION EFFICIENCY ($\eta$) FOR STATIC FILES

| Algorithm | Encryption Time (s) | CPU (%) | $\eta$ (%) |
|---|---|---|---|
| AES-CBC | 0.53 | 35.0 | 91.73 |
| AES-GCM | 0.46 | 40.0 | 85.50 |
| ChaCha20 | 0.61 | 38.0 | 79.94 |
| Blowfish | 0.67 | 43.0 | 77.21 |
| Triple DES | 1.98 | 72.0 | <20 |

The results show that AES-CBC achieves the highest $\eta$ (91.73%), confirming its suitability for structured content. It exhibits a well-balanced trade-off between speed, resource consumption, and minimal ciphertext expansion (≈ 2%). AES- GCM, while faster in some cases, incurs slightly higher over- head due to its built-in authentication tag. ChaCha20, despite being highly efficient for stream data, performed slightly below AES-CBC due to setup time and nonce overhead in small files. Blowfish and Triple DES performed significantly worse, with Triple DES showing minimal throughput and excessive CPU usage, leading to an $\eta$ below 20%.

### B. Streamed File Types: Audio and Video

Table II presents result for audio and video files, represent- ing large, continuous, and often compressed data formats.

TABLE II
Encryption Efficiency ($\eta$) for Audio and Video Files

| Algorithm | Encryption Time (s) | CPU (%) | $\eta$ (%) |
|---|---|---|---|
| ChaCha20 | 0.61 | 38.0 | 96.88 |
| AES-GCM | 0.46 | 40.0 | 85.50 |
| AES-CBC | 0.53 | 35.0 | 58.77 |
| Blowfish | 0.67 | 43.0 | 77.21 |
| Triple DES | 1.98 | 72.0 | <20 |

ChaCha20 clearly outperforms other algorithms with an effi- ciency score of 96.88%. Its stream-oriented architecture avoids block padding and initialization complexity, resulting in rapid processing and negligible ciphertext growth. AES-GCM, while secure and authenticated, consumed more CPU and added encryption overhead, reducing its overall efficiency. AES- CBC, when applied to video files, showed longer processing times and higher memory usage due to block-based buffering, yielding an efficiency of 58.77%. Blowfish and Triple DES again performed poorly, with Triple DES lagging far behind due to its outdated design and computational inefficiency [13].

### C. Summary

These results support the central hypothesis of this work: that intelligent algorithm selection based on file type yields superior overall system performance. AES-CBC is confirmed as the most efficient and secure choice for static data encryp- tion, while ChaCha20 is validated as the optimal cipher for high-volume, streaming content such as audio and video.

In practical deployments, AES-GCM remains the stan- dard for authenticated encryption when hardware accelera- tion (AES-NI) is available. However, in pure software envi- ronments, ChaCha20 achieves superior speed, especially for large and continuous data streams, validating its selection for audio and video files in our framework. Blowfish, while still occasionally used, lacks the modern key size recommendations and is not preferred for long-term secure storage. Triple DES, despite its historical importance, is largely deprecated due to weak effective key size and slow performance.

Our adaptive approach therefore achieves an optimal bal- ance:

- AES-CBC for text and image data ensures compatibility, strong diffusion, and cryptographic maturity.
- ChaCha20 for audio and video data leverages stream ci- pher advantages, achieving high speed and timing safety. Overall, the results demonstrate that careful file-type-based cipher selection significantly improves encryption efficiency without sacrificing security.

## VI. CONCLUSION

This work presented a comprehensive adaptive encryption framework capable of dynamically selecting encryption algo- rithms based on file type characteristics. The system integrates AES-CBC for static structured data such as text and images, and ChaCha20 for unstructured streaming media such as audio and video. It further enhances security through dynamic key derivation using PBKDF2 and user-specific salts, ensuring strong resistance against brute-force attacks. Both the Tkinter- based local application and the Flask-based web application successfully provide flexible encryption solutions tailored for different user environments.

Performance evaluation validates the system's effective- ness: ChaCha20 offers substantial improvements in encryp- tion speed for media files, while AES-CBC maintains ro- bust security for documents and images. The comparative analysis against alternative encryption algorithms shows that the selected ciphers strike a strong balance between security, performance, and deployment simplicity.

By combining intelligent cipher selection, flexible deploy- ment models, and modern cryptographic practices, the frame- work addresses the critical challenge of securing heteroge- neous data with minimal user friction and computational overhead.

## Future Work

While the current framework achieves significantly improve- ments in adaptive multimedia encryption, several future en- hancements are envisioned:

1) Integration of Authenticated Encryption: Upgrading the encryption modes to AES-GCM and ChaCha20-Poly1305 would provide built-in data integrity and authentication, re- ducing the need for external hash validation mechanisms.

2) Scrypt-Based Key Derivation: Adding support for Scrypt alongside PBKDF2 would offer users stronger, memory-hard password-based key derivation options, further hardening re- sistance against brute-force and dictionary attacks.

3) Compression Layer: Incorporating an intelligent com- pression module before encryption would optimize storage and transmission costs, particularly for text-heavy or redundant data formats.

4) Real-Time Streaming Encryption: Extending the system to encrypt live audio and video streams dynamically without significant latency, suitable for conferencing and real-time communication applications.

5) Hardware Acceleration Support: Adapting the system to utilize hardware cryptographic modules (e.g., AES-NI, TPMs) and GPU acceleration would boost performance on capable devices.

6) Distributed and Cloud Integration: Expanding the web application with cloud storage backends and distributed en- cryption/decryption pipelines would enable secure storage and sharing of encrypted content across geographically distributed environments.

Through these enhancements, the adaptive encryption sys- tem can evolve into a highly scalable, intelligent, and resilient platform for secure multimedia management in diverse appli- cation domains.

## REFERENCES

[1] R. Doomun, J. Doma and S. Tengur," AES-CBC Software Execution Optimization," 2008 International Symposium on Information Tech- nology, Kuala Lumpur, Malaysia, 2008, pp. 1-8, doi: 10.1109/IT- SIM.2008.4631586.

[2] A. Maolood, E. Gbashi and E. Shakir," Novel lightweight video encryption method based on ChaCha20 stream cipher and hybrid chaotic map," International Journal of Electrical and Computer En- gineering (IJECE), vol. 12, no. 5, pp. 4988–5000, Oct. 2022, doi: 10.11591/ijece. v12i5.pp4988-5000.

[3] A. Wattar," A new approach for the image encryption using AES cipher in ECB mode," Turkish Journal of Computer and Mathematics Education (TURCOMAT), vol. 14, no. 3, pp. 1039–1052, Jul. 2020, doi: 10.61841/turcomat. v14i03.14117.

[4] A. Iuorio and A. Visconti," Understanding Optimizations and Measuring Performances of PBKDF2," in Advances in Intelligent Systems and Computing, vol. 1015, pp. 101–114, Jan. 2019, doi: 10.1007/978-3-030-11437-4_8.

[5] A. A. Muhammed, A. T. Al-Dulaimy, and A. S. Al-Badrany, "A comparative analysis of AES, Blowfish, Twofish, Salsa20, and ChaCha20 encryption algorithms for image security," *arXiv preprint arXiv:2407.16274*, 2024.

[6] A. Elbadawi and A. Hadi, "AES-CBC software execution optimization for wireless sensor devices," *International Journal of Engineering Research and Applications*, vol. 10, no. 3, pp. 21–27, 2020.

[7] R. Sharma and S. Sabharwal, "High-definition image encryption algo- rithm based on AES modification," *International Journal of Engineering Research and Technology (IJERT)*, vol. 9, no. 12, pp. 2278–2284, 2020.

[8] S. Rani and R. Saravanan, "An image encryption method based on chaos system and AES algorithm," *International Journal of Engineering

Trends and Technology (IJETT)*, vol. 69, no. 5, pp. 213–220, 2021.

[9] O. S. Faragallah, A. I. Sallam, and H. S. El-Sayed, “Utilization of HEVC ChaCha20-based selective encryption for secure telehealth video conferencing,” *Computers, Materials Continua*, vol. 70, no. 1, pp. 831–845, 2022.

[10] M. Al-Khafaji and R. Hussein, “Performance evaluation of authentication-encryption and confidentiality block cipher modes of operation on digital image,” *International Journal of Computer Applications*, vol. 182, no. 4, pp. 1–7, 2018.

[11] P. Torkaman and Z. Eslami, “A revision of a new chaos-based image encryption system: Weaknesses and limitations,” *Multimedia Tools and Applications*, vol. 81, pp. 2045–2067, 2022.

[12] S. R. R. Sabarish and P. Sakthi Murugan,” Cryptographer: Encryption & Decryption Tool,” International Journal of Research Publication and Reviews, vol. 6, no. 3, pp. 5078–5083, Mar. 2025. [Online]. Available: https://ijrpr.com/uploads/V6ISSUE3/IJRPR40321.pdf

[13] B. A. Buhari, H. Abdulkadir, R. Sulaiman and S. Shehu,” Performance and Security Analysis of Symmetric Data Encryption Algorithms: AES, 3DES and Blowfish,” ResearchGate, Jan. 2025. [Online].