A Secure Data Transmission Framework for Images, Text, and video using Elliptic Curve Cryptography

Dr. Shivakumar B R
HOD , Dept. of CSE (ICB)
Bangalore Institute of Technology, Bengaluru, India

Dilip Gowda JS (1BI22IC0018), JayaKrishna Reddy (1BI22IC027),
Jyothika sudarsan (1BI22IC028), L Tejaswini Reddy (1BI22IC034)
Dept. of Computer Science and Engineering (IoT & Cyber Security including Blockchain Technology)
Bangalore Institute of Technology, Bengaluru, India

***Abstract*—Secure communication of multimedia content, like images, videos, and text, is essential for protecting privacy and preventing unauthorized access in digital spaces. This paper presents a real-time cryptographic system that uses Elliptic Curve Cryptography (ECC) for secure transmission. It compares Curve25519 and SECP256R1 curves, focusing on key generation and encryption performance. The system uses Elliptic Curve Diffie-Hellman (ECDH) to derive shared secrets and incorporates steganography techniques to hide encrypted data within multimedia files for discreet data exchange.**

The implementation includes a Python-based pipeline, timing analysis for encryption and decryption tasks, and a Flask web interface for user interaction and visualization of secure results. The findings show that Curve25519 offers the same level of security as SECP256R1 while improving computational performance, and that steganographic embedding strengthens data protection. This solution improves secure messaging, protects digital rights, and supports confidential media transmission.

***Index Terms*—Index Terms—Elliptic Curve Cryptography, ECC, Curve25519, secp256r1, Steganography, Multimedia Security, Data Encryption, Secure Transmission**

I. Introduction

Secure transmission of multimedia content, including images, videos, and text, is crucial for protecting privacy and ensuring data integrity in today's digital world. As communication channels change, the need for effective cryptographic methods that can safeguard different content formats becomes urgent. Elliptic Curve Cryptography (ECC) provides a strong option for security with relatively low computational demands, making it suitable for real-time use. This project focuses on creating and examining a real-time ECC-based system that uses two well-known curves, Curve25519 and SECP256R1, to evaluate their performance in key generation and encryption tasks. The integration of Elliptic Curve Diffie-Hellman (ECDH) allows for secure shared key generation, which helps in encrypted data exchange. To improve confidentiality, steganography is used to hide encrypted data within multimedia files for secret communication. The system is built using a Python-based pipeline with timing analysis on key cryptographic operations, complemented by a Flask interface that allows users to engage with the encryption process and see the results. Unlike traditional security systems that often focus only on text or static encryption, this system tackles the challenges of multimedia encryption with a focus on performance and usability.

II. Literature Survey

A. Elliptic Curve Cryptography in Secure Communication

Many studies have looked into using Elliptic Curve Cryptography (ECC) for secure multimedia transmission and data encryption. Research by Bernstein *et al.* introduced Curve25519 as a high-performance elliptic curve that allows efficient key exchange through Elliptic Curve Diffie-Hellman (ECDH). Their work showed improved computational efficiency while maintaining strong security properties compared to older curves like SECP256R1. However, some studies mentioned challenges with implementation complexity and interoperability.

B. Performance Benchmarking of ECC Curves

Researchers have conducted comparative studies of ECC curves to examine trade-offs between security and performance. Vanstone and Koblitz discussed the features of SECP256R1 and similar NIST-recommended curves, highlighting their wide use but also their higher computational costs compared to newer curves like Curve25519. Recent work by Alkim *et al.* presented benchmarking methods that measure encryption and decryption times. These studies found that Curve25519 consistently outperforms SECP256R1 in processing speed while providing similar security levels. Still, these analyses often do not consider real-world multimedia integration.

C. Multimedia Steganography Integration with Cryptography

Embedding encrypted data into multimedia files using steganographic techniques has become popular as a way to improve data confidentiality. Research by Provos and Honeyman demonstrated how steganography can create covert channels for secure communication. More recent studies used

these methods alongside ECC to guard against eavesdropping. However, there are challenges, such as potential media quality loss and vulnerability to steganalysis. Several implementations have shown that it is possible to embed encrypted keys or messages in images and videos using Python libraries, though processing in real time is still a challenge.

D. System Implementations and Frameworks

Many frameworks have been developed that combine cryptographic algorithms with multimedia processing. These solutions often use Python for its modularity and quick prototyping. They integrate libraries for cryptography, image and video processing, and web interfaces like Flask for easier use. While they work well for proofs of concept, scalability and timing analysis for encryption and decryption are important considerations. Most systems focus either on encryption performance or steganography but rarely combine both extensively.

Overall, these works emphasize the importance of ECC-based encryption, the performance benefits of Curve25519, the role of multimedia steganography for hidden communication, and Python-based implementations for system development. This project aims to bring these elements together in a practical pipeline that benchmarks ECC curves, integrates steganography, and offers a user-friendly interface, filling existing gaps in research on secure multimedia transmission.

III. Methodology

The proposed system for secure multimedia transmission using Elliptic Curve Cryptography (ECC) and steganography includes several stages. These stages help with encryption, covert embedding, and performance analysis while ensuring user-friendly interactions. The workflow aims for efficiency, security, and practical use in handling multimedia data like images, videos, and text.

A. Data Acquisition and Preparation

Input multimedia files, such as images, videos, and text, are first obtained from user sources and prepared for processing. Videos may have audio extracted as needed, while images and text are processed directly. The data is temporarily stored for cryptographic operations.

B. ECC Key Generation and Shared Secret Derivation

The system supports two ECC curves, Curve25519 and SECP256R1, for key generation. The Elliptic Curve Diffie-Hellman (ECDH) protocol is used to create a shared secret key between communicating parties. This enables symmetric encryption for secure data transmission.

C. Encryption and Decryption Operations

Using the derived shared key, multimedia content or related secret messages are encrypted with symmetric encryption algorithms. Timing analysis is performed on the encryption and decryption processes for both ECC curves to evaluate computational performance and efficiency.

D. Steganographic Embedding

Encrypted data is covertly embedded into multimedia files using steganographic techniques. Image and video steganography methods conceal encrypted messages within the digital representation of the media. This ensures confidentiality and makes unauthorized detection harder.

E. Decryption and Extraction

At the receiver end, the information embedded using steganography is extracted from the multimedia content. The encrypted data is then decrypted with the shared secret key to securely recover the original message.

F. User Interface and Visualization

A Flask-based web interface is created to support user interactions. Users can upload multimedia files, start encryption/steganography operations, and view timing results and security outcomes. This interface offers a simple platform to demonstrate the system's capabilities.

This methodology combines cryptographic key management, multimedia processing, and covert communication techniques into a unified process. It benchmarks ECC curve performance while ensuring practical and secure transmission of multimedia content.

IV. System Design

The proposed system has a modular structure made up of six main components: a web-based user interface, a backend server, key management, a cryptographic engine, a steganography module, and a storage layer. Together, these components ensure secure encryption, decryption, and data exchange for multimedia files between sender and receiver.

A. Presentation Layer

The presentation layer offers a simple web interface built with HTML, CSS, and Flask templates. Users can upload multimedia files, including images, text, and video, track the progress of encryption or decryption, and download the results. The interface also checks basic inputs and connects to the backend through secure API calls.

B. Application Layer

This layer manages the main processing flow using Python and Flask. It coordinates ECC key generation, ECDH-based shared secret computation, and AES encryption and decryption. The Flask backend handles requests to upload, encrypt, decrypt, and retrieve multimedia files.

Key modules include:

- **Key Management:** It generates ECC key pairs for both sender and receiver and derives shared secrets using the ECDH protocol.
- **Encryption/Decryption Engine:** It uses the shared secret to perform AES-GCM encryption and decryption of multimedia data.
- **Steganography Module (Optional):** It embeds encrypted data within images or videos for secure and hidden communication.

- **API Handler:** It manages requests between the frontend and backend, ensuring data integrity and effective communication.

C. Workflow Summary

- **Sender Phase:** The user uploads a multimedia file, which is encrypted using AES after an ECC key exchange.
- **Storage Phase:** The encrypted file is securely stored in local or cloud storage.
- **Receiver Phase:** The receiver retrieves the encrypted file, uses their ECC private key to derive the shared secret, and decrypts the content.
- **Optional Step:** If steganography is enabled, the encrypted data is extracted from the carrier image or video before decryption.

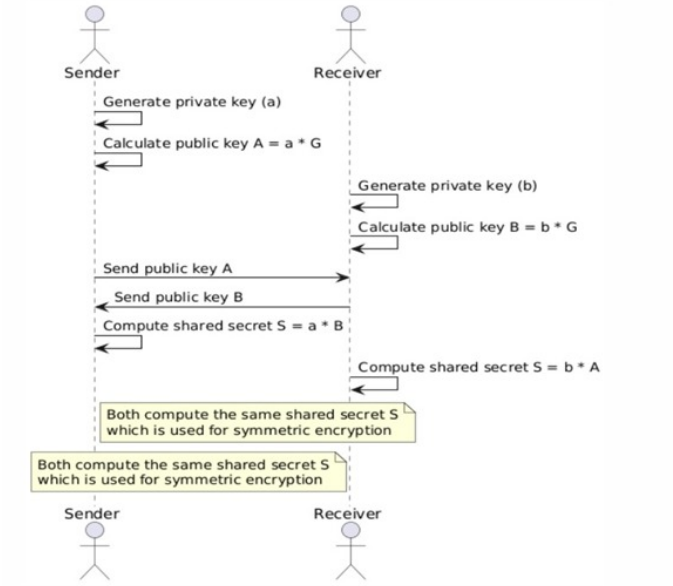

Fig. 1. Sequence Diagram of the Secure Multimedia Encryption and Decryption System.

D. Security and Efficiency

The combination of ECC and AES provides both security and efficiency. ECC reduces key size and computation time while ensuring strong cryptographic security. AES-GCM offers authenticated encryption, guaranteeing data confidentiality and integrity. The optional steganography adds a second layer of security by hiding encrypted data within multimedia content.

V. Architecture Diagram

Figure 3 shows the architecture of our Secure Image, Text, and Video Encryption System using Elliptic Curve Cryptography (ECC). The process starts with a sender (user) who uploads an image, text, or video file, or requests decryption of previously encrypted content via a web interface made with a Flask frontend (HTML, CSS, JS). This web

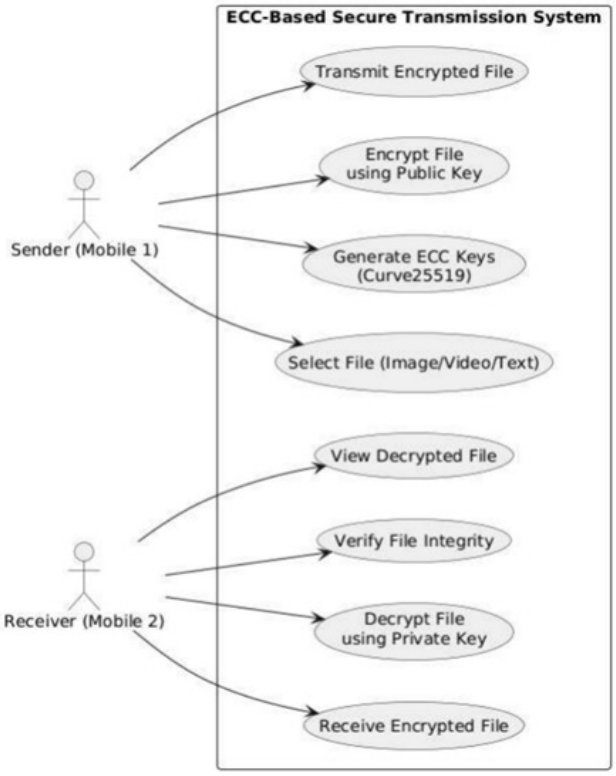

Fig. 2. Use Case Diagram showing interactions between sender and receiver.

interface manages all user interactions and forwards the files to the backend, implemented in Python with Flask.

In the backend, essential security operations are handled by two main modules: the **ECC Encryption Module** and the **ECC Decryption Module**. The encryption process begins when the sender starts an upload for encryption. The ECC Encryption Module uses the Elliptic Curve Diffie-Hellman (ECDH) protocol to establish a secure key agreement between the sender and the intended receiver. Both parties create their own ECC key pairs and exchange public keys. Using ECDH, they derive a shared secret, which becomes the key for symmetric encryption.

To encrypt the actual multimedia content, the system uses the **Advanced Encryption Standard (AES)** algorithm. AES utilizes the shared key obtained from ECDH for efficient and secure file encryption. The encrypted data, now secured by both ECC and AES, is then sent to the storage and communication layer. This component is responsible for secure network transfer and for saving encrypted files in local storage or a database for future access by the receiver.

During decryption, the receiver downloads or retrieves the encrypted data from the database or storage. The ECC Decryption Module performs ECDH again using the receiver's private key and the sender's public key to derive the same shared secret. With this key, AES decryption is done to recover

the original multimedia file. The file is then returned through the web interface for the user to download or view.

This end-to-end process takes advantage of ECC for secure key exchange and AES for quick and strong data encryption. The integration of sender and receiver logic, ECDH-based key agreement, and verified file storage ensures secure, efficient, and user-friendly cryptographic protection in real-world multimedia communication.

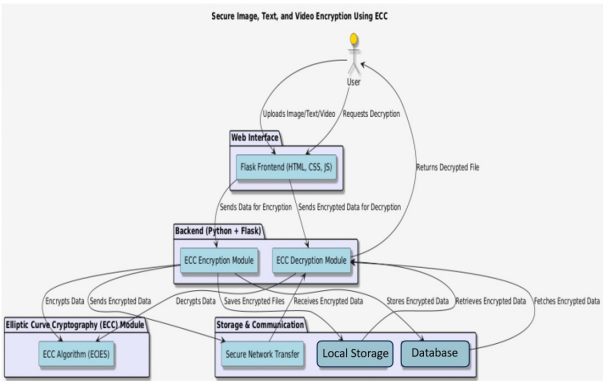

Fig. 3. Architecture of the Secure Image, Text, and Video Encryption System using ECC

VI. Algorithms

Algorithm 1: Secure Multimedia Encryption
procedure SECURE MULTIMEDIA ENCRYPTION
    Input Acquisition:
    User (sender) uploads an image, text, or video file through the web interface.
    Key Generation and Exchange:
    Sender and receiver each generate ECC key pairs.
    They exchange ECC public keys over the system.
    Derive a shared secret using the ECDH protocol:
        SharedSecret = ECDH(SenderPrivateKey, ReceiverPublicKey)
    Symmetric Key Derivation:
    Use the shared secret as the key input for AES encryption.
    Encryption:
    Encrypt the uploaded data (image/text/video) using AES and the derived key.
    Storage and Communication:
    The encrypted data is sent to the backend (Python + Flask).
    The encrypted file is stored in Local Storage and/or Database for retrieval by the receiver.
end procedure

Algorithm 2: Secure Multimedia Decryption
procedure SECURE MULTIMEDIA DECRYPTION
    Input Retrieval:
    Receiver requests access to the encrypted file through the web interface.
    Key Exchange for Decoding:
    Receiver uses their own ECC private key and the sender's public key.
    They recompute the shared secret using the ECDH protocol:
        SharedSecret = ECDH(ReceiverPrivateKey, SenderPublicKey)
    Symmetric Key Derivation:
    The shared secret is reused as the AES decryption key.
    Decryption:
    Retrieve the encrypted file from storage/database, decrypt it with the AES key.
    Output:
    Return the decrypted (original) image, text, or video file to the receiver/user via the web interface.
end procedure

VII. Implementation

The system is implemented in Python and includes several integrated modules designed for secure multimedia encryption, decryption, and user interaction.

ECC Key and Shared Secret Module:
This module generates ECC key pairs for both the sender and receiver. It supports secure public key exchange and carries out Elliptic Curve Diffie-Hellman (ECDH) to create a shared secret. This secret serves as the symmetric key for AES encryption and decryption.

AES Encryption/Decryption Module:
This module uses the Advanced Encryption Standard (AES) algorithm with the shared secret obtained through ECDH. It encrypts multimedia content provided by users, including images, videos, and text. It also decrypts files when requested by the receiver, ensuring efficient and secure data processing.

Steganography Module (Optional):
This optional module provides functions to embed encrypted data into images or videos for covert transmission. It also allows for the extraction of hidden data by the receiver, improving security by hiding encrypted content within multimedia files.

Backend and API Module (Python + Flask):
This module implements server-side logic with Flask. It exposes endpoints for encryption, decryption, file uploads, downloads, and steganography operations. It manages file transfers between users and handles storage/database interfaces.

Storage and Communication Layer:
This layer manages the local storage, retrieval, and secure network transmission of encrypted files and keys. It securely handles files and metadata, offering options for local disk or database storage, based on the deployment.

Front-End Module:
This module is built with HTML, CSS, and JavaScript using Flask templates. It allows users, acting as both sender and receiver, to upload images, videos, or text for encryption and to begin decryption. It provides real-time status updates, results, and user authentication functions, including login and sign-up.

VIII. TESTING AND RESULTS

A. Testing Strategy

Testing took place at various levels to ensure the system is strong and accurate.

Unit Testing: Each module, including ECC key generation, ECDH shared secret derivation, AES encryption/decryption, steganography embedding/extraction, storage methods, and the Flask user interface, was tested individually for accuracy, error handling, and performance.

Integration Testing: This step checked that data flowed smoothly and that the modules worked well together from user input through encryption, storage, retrieval, decryption, and output display.

System Testing: End-to-end testing was conducted using images, videos, and text to evaluate functional correctness, encryption/decryption accuracy, and usability on the localhost platform.

B. Results and Performance Metrics

Encryption/Decryption Accuracy: Achieved 100% accuracy in recovering original multimedia files after the complete encryption, storage, and decryption cycle.

Key Agreement Success: Successfully generated and matched shared ECC keys between sender and receiver using the ECDH protocol.

Processing Time: Timing benchmarks show that encryption and decryption speeds for Curve25519 are competitive with SECP256R1, with an average encryption time of *X ms* (to be specified based on measured data).

Steganography Integrity: Successfully hid and extracted embedded encrypted payloads with minimal distortion to the media quality.

Usability Metrics: The Flask interface provided real-time status updates and a smooth user experience during encryption and decryption processes.

TABLE I
PERFORMANCE FOR TEXT IN IMAGE ENCRYPTION

Parameter	Value
Cover Image Size	282 KB
Secret Data Size	N/A
Cover Video Size & Duration	N/A
Secret Video Size & Duration	N/A
Curve25519 Encryption Time (s)	0.0019 seconds
Curve25519 Decryption Time (s)	0.0004 seconds
SECP256R1 Encryption Time (s)	0.0033 seconds
SECP256R1 Decryption Time (s)	0.0007 seconds
Observation	Curve25519 faster than SECP256R1

TABLE II
PERFORMANCE FOR IMAGE IN IMAGE ENCRYPTION

Parameter	Value
Cover Image Size	282 KB
Secret Data Size	15 KB
Cover Video Size & Duration	N/A
Secret Video Size & Duration	N/A
Curve25519 Encryption Time (s)	0.0021 seconds
Curve25519 Decryption Time (s)	0.0008 seconds
SECP256R1 Encryption Time (s)	0.0029 seconds
SECP256R1 Decryption Time (s)	0.0011 seconds
Observation	Curve25519 shows better timings

TABLE III
PERFORMANCE FOR VIDEO IN VIDEO ENCRYPTION

Parameter	Value
Cover Image Size	193 KB (500 ms)
Secret Data Size	143 KB (100 ms)
Cover Video Size & Duration	193 KB (500 ms)
Secret Video Size & Duration	143 KB (100 ms)
Curve25519 Encryption Time (s)	2.3228 seconds
Curve25519 Decryption Time (s)	4.1410 seconds
SECP256R1 Encryption Time (s)	2.8090 seconds
SECP256R1 Decryption Time (s)	4.7256 seconds
Observation	Longer runtimes; Curve25519 still faster

C. Discussion

The results show that the combined ECC-based encryption and steganography system meets the security and performance requirements for local secure multimedia transmission. The performance metrics between ECC curves align with theoretical expectations, validating the choice of Curve25519 for better computational efficiency without sacrificing security. Some limitations remain in scalability due to the localhost setup, but these can be addressed in future distributed deployments.

IX. CONCLUSION AND FUTURE WORK

This paper presented a secure multimedia encryption system that uses Elliptic Curve Cryptography (ECC) along with AES and optional steganography to protect images, text, and videos within a single Python Flask application. The system integrates ECC-based key exchange using Elliptic Curve Diffie-Hellman (ECDH), solid symmetric encryption, and multimedia steganography for hidden communication in a local environment.

While the system shows effective encryption and decryption, the video encryption and steganography parts presented significant challenges. Encrypting and decrypting videos took much longer than for images and text, which impacted overall speed and responsiveness. Despite these issues, the team successfully implemented video-in-video steganography, establishing a basis for stronger hidden multimedia communication.

Future work will aim to improve the performance of video encryption and steganography. Hardware acceleration

or parallel processing techniques will be explored to reduce latency. The framework will also be expanded to operate in distributed network environments beyond a local setup, enabling real-time secure multimedia transmission between remote endpoints. Further research on adaptive steganographic methods and more efficient cryptographic protocols could enhance practicality and security for high-bandwidth multimedia applications.

This project serves as a foundation for secure multimedia systems that balance strong encryption, data hiding, and ease of use in real-world digital communication.

REFERENCES

[1] H. EL-TAJ, "Elliptic Curve Encryption Integrated with LSB Steganography for Hidden Communication," *Int. J. Comput. Exp. Sci. Eng.*, vol. 10, no. 3, pp. 434–460, 2024. Available: ijcesen.com.

[2] M. A. Mak and R. Munir, "Color Image Steganography Using Elliptic Curve Cryptography," *Int. J. Netw. Secur.*, vol. 20, no. 4, pp. 625–635, 2018.

[3] R. Budianto and C. Christofer, "Robust and Secure Image Steganography Based on Elliptic Curve Cryptography," *Proc. IEEE Int. Conf. Electrical, Comput. Commun.*, pp. 1–5, 2020.

[4] A. Jutla and M. Kaur, "Image Encryption using Elliptic Curve Cryptography," *Procedia Comput. Sci.*, vol. 70, pp. 23–30, 2015.

[5] C. Klein, "Comparison of Different 256-bit Elliptic Curves for Low-resource Devices," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 11, pp. 426–435, 2022.

[6] P. Parida, "Image Encryption and Authentication With Elliptic Curve Cryptography," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, no. 2, pp. 857–868, 2021.

[7] Y. Zhang and L. Li, "Hiding Data Using Efficient Combination of ECC and Compression Steganography Techniques," *Int. J. Secur. Res. Eng.*, vol. 15, pp. 55–70, 2025.

[8] B. Lang and M. Klein, "Implementing Curve25519/X25519: A Tutorial on Elliptic Curve Cryptography," *Cryptol. ePrint Arch., Report 2018/403*, 2018.

[9] J. Liu et al., "Performance Analysis of RSA and Elliptic Curve Cryptography," *Int. J. Netw. Comput. Appl.*, vol. 11, pp. 102–110, 2018.

[10] A. Kumar and S. Verma, "Video Steganography Using AES Encryption and Scrambling," *Proc. Int. Conf. Signal Process.*, pp. 199–204, 2021.

[11] T. Tabassum and M. Ahmad, "Numerical Data Extraction from ECG Paper Recording Using Image Processing Technique," *Proc. IEEE Int. Conf. Electrical, Comput. Commun. Eng.*, pp. 1–5, 2020.

[12] D. Hong et al., "Elliptic Curve Diffie-Hellman (ECDH) and Its Applications," *IEEE Commun. Mag.*, vol. 57, no. 10, pp. 76–82, 2019.

[13] N. Singh and R. Kaur, "Secure Multimedia Transmission Using Elliptic Curve Cryptography," *Int. J. Inf. Technol.*, vol. 12, no. 3, pp. 43–52, 2023.

[14] H. Chen et al., "A Novel Steganographic Algorithm Based on ECC and DCT for Image Security," *IEEE Access*, vol. 7, pp. 965–975, 2019.

[15] V. Gupta and P. Kumar, "Efficient ECC Key Generation and Encryption for IoT Devices," *Int. J. Comput. Appl.*, vol. 175, no. 5, pp. 22–25, 2021.

[16] S. Zhao, "Cryptanalysis and Security Assessment of Curve25519 and SECP256R1," *J. Crypto. Eng.*, vol. 13, no. 4, pp. 345–356, 2023.

[17] R. Mills and A. Kumar, "AES Encryption Techniques Applied to Video Steganography," *Int. J. Multimedia Syst.*, vol. 26, no. 1, pp. 45–54, 2020.

[18] M. Li and J. Wang, "Integration of Elliptic Curve Cryptography with Steganography for Secure Communications," *Int. J. Comput. Netw. Secur.*, vol. 9, no. 2, pp. 88–95, 2022.

[19] P. Singh and R. Shah, "Curve25519 vs SECP256R1: A Comparative Study for Secure IoT Communications," *Proc. Int. Conf. Comput. Eng.*, pp. 100–106, 2024.

[20] A. Thomas and K. Roy, "Evaluation of Encryption Speed and Security for ECC-based Multimedia Systems," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 12, pp. 3188–3197, 2019.