Philippine Journal of Interdisciplinary Research (PJIR)
Volume I, Issue 2 (August 2025)

Enhanced AES Cryptography Algorithm for Secured Health Information Exchange

Mary M. Asia[1]

[1]President Ramon Magsaysay State University

Corresponding email: maryasia008@gmail.com
Received: 18 July 2024; Accepted: 13 August 2024; Available online: 30 October 2025

Abstract. Globally, the healthcare industry is a critical sector that directly influences human life. Ensuring the confidentiality, integrity, and authenticity of health data is paramount to protecting individual privacy. Although the Advanced Encryption Standard (AES) is a widely recognized encryption technique, it has inherent vulnerabilities, particularly in secure key sharing. Compromises in these channels can undermine the overall strength of AES encryption. In response to the increasing threat of data breaches, numerous cryptographic algorithms have been developed to protect digital health records and communication. These include symmetric algorithms, such as the Advanced Encryption Standard (AES) and Data Encryption Standard (DES), and asymmetric algorithms, such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC). This paper presents an enhanced AES algorithm integrated with an Elliptic Curve Diffie-Hellman (ECDH), which strengthens key management by offering secure key generation and additional cryptographic layers. The research employed an experimental design utilizing PyCryptodome for implementation, alongside tools such as NumPy, psutil, and Matplotlib for performance testing and analysis. Comparative evaluations of the enhanced AES-ECDH and standard AES algorithms were conducted in terms of execution time, CPU usage, memory consumption, and security analysis. Dummy datasets were used to uphold ethical standards, ensuring that sensitive information was not compromised during testing.

The findings revealed that while the enhanced AES-ECDH algorithm significantly improves security by offering features such as forward secrecy and heightened resistance to various attacks, it comes at the expense of increased resource consumption. Despite this trade-off, the enhanced algorithm is highly suitable for scenarios that prioritize data protection over system performance, particularly in healthcare environments.

Keywords: *AES, ECDH, cryptography, security, encryption, decryption, key exchange*

INTRODUCTION

The health industry plays a vital role in ensuring the security of data in different sectors. With the widespread digitalization of medical services and increasing reliance on

electronic health records (EHRs), ensuring the secure exchange of sensitive health information among healthcare providers, patients, and authorized entities has become essential. Protecting the confidentiality, integrity, and authenticity of heath data is critical not only to uphold patient trust but also to comply with international data privacy standards like the Health Insurance Portability and Accountability Act (HIPAA) in the United States (Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, 2001).

In response to the rising threat of data breaches, several cryptographic algorithms have been developed to safeguard digital health records and communication. This includes symmetric algorithms like Advanced Encryption Standard (AES) and Data Encryption Standard (DES) and asymmetric algorithms like RSA and Elliptic Curve Cryptography (ECC). The choice of algorithm played an important role in balancing security and efficiency. An algorithm such as the Advanced Encryption Standard (AES) was preferred to secure data transmission. The AES algorithm was developed by two Belgian cryptographers, Vincent Rijmen and Joan Daemen, through a process initiated by NIST to replace the outdated Data Encryption Standard (DES) and to meet the growing security needs of the 21st century (Brahmaiah et al., 2023). Unlike DES, AES operates on a substitution-permutation network design principle, enhancing its efficiency in both software and hardware applications (Al-Khafaji & Abdul, 2022). It was the best algorithm recognized by NIST and approved by NSA for protection of the top-secret information and national security systems (National Institute of Standards and Technology, 2001).

However, given the ever-growing computational power available to attackers, continuously enhancing AES security features is imperative. In addition, AES's key sharing of AES depends on a secure channel; if that channel is compromised, the security of the information is compromised (Bhowmika et al., 2022). Therefore, it is necessary to secure the key to the AES algorithm. To address these concerns, this research enhanced AES cryptography by integrating Elliptic Curve Diffie-Hellman (ECDH) Key Exchange and additional round keys in both the pre-processing and post-processing stages.

To overcome these limitations, hybrid encryption techniques are becoming popular where AES is combined with Elliptic Curve Diffie-Hellman (ECDH) augmenting security because they enable secure key exchange even over insecure channels. ECDH offers forward secrecy and a very high level of security, even while keeping key sizes small compared to RSA, making ECDH an ideal candidate for health applications where performance meets security requirements. This study proposes an Enhanced AES cryptography algorithm that integrates ECDH for secure key generation and includes the use of additional round keys in both the pre-processing and post-processing phases to enhance resilience against advanced attacks.

The performance of the proposed algorithm was evaluated on parameters such as execution time, CPU utilization, memory usage, and security, and compared to standard AES implementations. A rigorous security analysis was conducted to validate its efficiency and effectiveness, offering insights into the relative strengths and weaknesses of the enhanced algorithm and to guide future developments in cryptographic protocols

for healthcare data security. This integration addresses the pressing need for enhanced security measures on health-information exchange platforms.

Literature Review

Cryptography plays a vital role in protecting digital information and ensuring the confidentiality, integrity, and authenticity of data across different sectors such as healthcare, finance, and e-commerce. This is done by encrypting and decrypting data using algorithms and cryptographic keys to keep it secure from unauthorized access and to allow secure communication (Raj & Kaur, 2023). Over time, cryptographic techniques have evolved considerably with the contributions of the mathematics and computing sciences. The more widely used algorithms developed are those of the Advanced Encryption Standard (AES), Rivest-Shamir-Adleman (RSA), Data Encryption Standard (DES), and Elliptic Curve Cryptography (ECC), which have served well against various security paradigms depending on the area of application context (Akram, 2022).

In the context of finance, however, encryption techniques are considerably useful, as they protect sensitive data, such as banking credentials, credit card numbers, and transaction history. RSA and ECC find prominent applications in secure online banking and digital signatures by providing high-level security using asymmetric encryption. Likewise, in the case of e-commerce, data encryption applying symmetric encryption methods, such as AES, provides a suitable and fast process for real-time transactions. SSL/TLS sits on top of previous methods and integrates them to provide an end-to-end secure channel for communication between clients and servers (Li, 2022). These real-world applications illustrate the versatility of cryptography and its critical role in fostering trust and reliability of digital services.

In the healthcare domain, the secure exchange of electronic health records (EHRs) and personal health information (PHI) has become increasingly important, particularly with the shift toward digital and cloud-based health systems. Traditional cryptographic techniques, such as DES and RSA, have been implemented in early health information systems, but their limitations in terms of speed and computational efficiency have prompted the exploration of advanced methods. AES then became the standard for encrypting medical data due to its speed and resistance against brute-force attacks (Carlet, Jakobovic, & Picek, 2021). However, the need for secure key exchange mechanisms has led to the exploration of ECC and its variants, such as the Elliptic Curve Diffie-Hellman (ECDH), which offers strong security with smaller key sizes, ideal for resource-constrained healthcare environments.

Hybrid encryption models that integrate symmetric and asymmetric techniques, such as AES with ECC or RSA, have gained popularity owing to their ability to combine the strengths of both methods. These hybrid approaches improve overall security by using asymmetric algorithms for key exchange and symmetric algorithms for data encryption. Studies by Negi, Shrestha, Borges, Sahana, & Das (2023), and Sharma, Kumar, & Gupta (2023) illustrates the effectiveness of hybrid techniques for data confidentiality and efficiency in health information systems. In addition, recent advancements, such as the

incorporation of ECDH into AES frameworks, have shown potential in minimizing computational load while ensuring robust protection, making them suitable for modern health IT infrastructures (Saepulrohman, Denih, Sukono, & Bon, 2020).

Continual innovation in cryptographic techniques, reflected in the refinement of existing algorithms and creation of hybrid models, underscores the dynamic nature of the field. Despite these advances, gaps remain in optimizing encryption for specific domains such as healthcare, where low latency, lightweight processing, and high security are simultaneously required. This study addresses this gap by proposing an enhanced AES algorithm integrated with ECDH for improved key management and overall system performance, contributing to ongoing efforts to develop secure and efficient solutions for health information exchange.

Conceptual Framework

Figure 1 Paradigm of the Study outlined the theoretical constructs and relationships that guided the development of an enhanced AES Cryptography Algorithm through the implementation of additional keys (pre-processing and post-processing) and ECDH key generation integration. The enhanced algorithm, referred to as AES-ECDH, combined two fundamental cryptographic techniques to improve both security and performance.

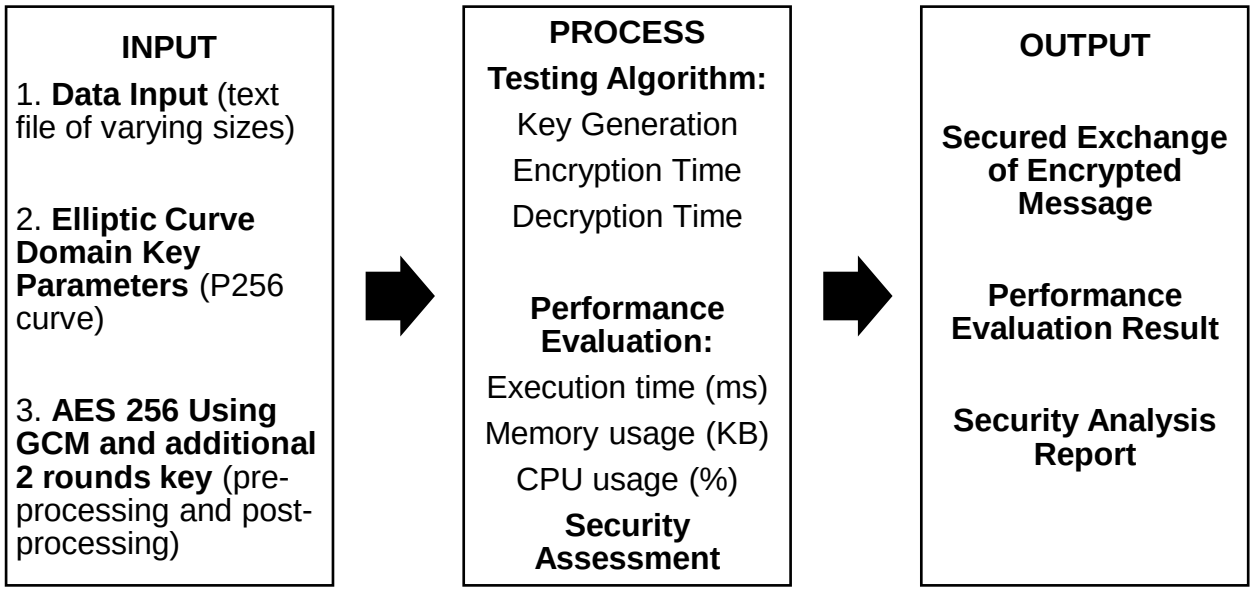

Figure 1. *Paradigm of the Study*

The conceptual framework of this study outlines the flow from input through processing to output, focusing on the integration of ECDH and an enhanced AES algorithm. The input phase includes data input from text files of varying sizes, elliptic curve domain key parameters based on the P256 curve, and AES 256 using the Galois/Counter Mode

(GCM) with additional rounds of key processing. These inputs are crucial for testing the robustness and scalability of the encryption algorithms.

This study involved several key activities in the process phase. The testing algorithm component included key generation using ECDH, measurement of encryption time, and decryption time. Performance evaluation assesses execution time in milliseconds, memory usage in megabytes, and CPU usage in percentage. In addition, a security assessment was conducted to analyze the security features and potential vulnerabilities of the encryption algorithm.

The output phase delivers several critical results, including secure exchange of encrypted messages and ensuring data confidentiality and integrity. The performance evaluation results provide detailed metrics of the execution time, memory usage, and CPU usage of the enhanced AES algorithm. Finally, the security analysis report offers a comprehensive assessment of the strengths of the algorithm and the potential areas for improvement. This structured approach aims to demonstrate the practical performance and theoretical security of the proposed cryptographic solution, highlighting its importance for securing sensitive health information.

METHOD

Research Design

The study used an experimental research design to evaluate the performance and efficiency of enhanced AES cryptography with integrated ECDH key generation and added key implementation for pre- and post-processing. Experimental research on cryptographic algorithms has played a crucial role in assessing security and performance, as observed in different studies. By systematically manipulating the variables and observing the outcomes, researchers have validated the functionality, efficiency, and security of cryptographic algorithms under various conditions. For instance, Johnson (2019) highlighted how controlled experiments help in understanding the practical performance and potential vulnerabilities of cryptographic methods. Similarly, Gupta, Singh, and Sharma (2021) emphasized that empirical testing could either support or challenge theoretical claims, contributing to the development of more robust cryptographic systems. Furthermore, Smith and Lee (2022) illustrated that experimental research provides valuable empirical evidence, ensuring that cryptographic algorithms meet security and efficiency standards in real-world applications.

The experimental framework involved examining the underlying mathematical concept of the AES-ECDH algorithm and demonstrating its cryptographic function. An experimental group utilized the proposed algorithm and compared its performance with that of a standard AES algorithm. This method includes a systematic exploration and analysis of existing concepts, hypothesis formulation, and experimental concept testing to deepen understanding and generate insights.

Instrument

To implement and evaluate the enhanced algorithm, Python Programming Language was utilized with PyCryptodome, a self-contained Python package of low-level cryptographic primitives that performs encryption and decryption operations and manages cryptographic keys. The development and testing phases were conducted using a range of specialized Python libraries.

Figure 2 presents a list of libraries utilized to ensure the accuracy, efficiency, and security of the algorithm.

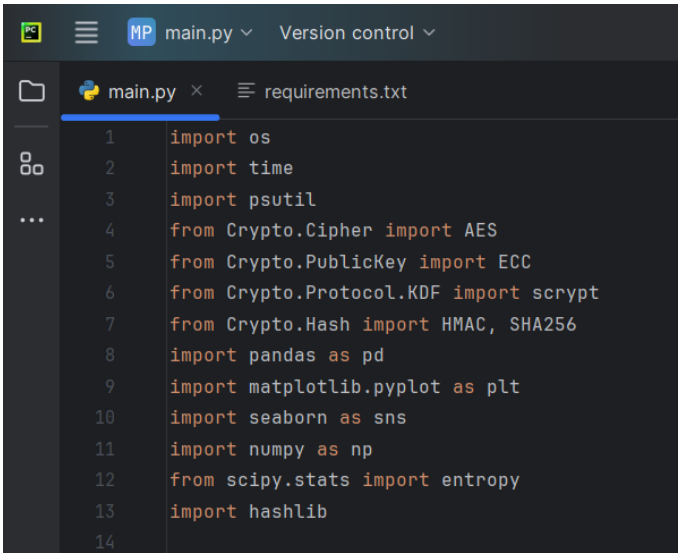

Figure 2. *Libraries Utilized for Implementing and Evaluating the Enhanced AES Algorithm*

To implement and evaluate the Enhanced AES Algorithm, a variety of Python libraries were utilized, each serving a specific purpose during the development process. The os and time modules are foundational in handling operating system-level tasks and timing functions, which are essential for evaluating performance metrics. The psutil library was instrumental in monitoring the system resource usage, such as CPU and memory, providing insights into the efficiency of the algorithm.

Cryptographic operations were central to this implementation, with the pycryptodome library as the primary tool. It included the Crypto.Cipher module for AES encryption, Crypto.PublicKey for handling Elliptic Curve Cryptography (ECC), Crypto.Protocol.KDF for key derivation using the scrypt algorithm, and Crypto.Hash for hashing functionalities such as HMAC and SHA-256. These cryptographic modules collectively enhance the security aspects of the AES algorithm by incorporating elliptic curve key exchange and robust hashing mechanisms.

For data manipulation and visualization, pandas were employed to manage and process datasets, whereas matplotlib and seaborn were used to create detailed plots and

visualizations that aid in analyzing the performance of the Enhanced AES Algorithm. Additionally, numpy is crucial for numerical computations, and scipy.stats is utilized for statistical analysis, specifically the entropy function, which measures the unpredictability and security of cryptographic keys. Finally, hashlib provided additional hashing functions, complementing cryptographic operations.

Data Collection

Generated random data with different file sizes were used to test the enhanced AES algorithm. Owing to the sensitive nature of healthcare data, most ethical considerations have been observed. Dummy data were used to mitigate risk during the testing phase. The execution of test cases involving key generation, data encryption, and data decryption was recorded. These results were used to measure the performance using metrics such as execution time, memory usage, CPU usage, and security. Additionally, a comparison of the performance with the standard AES algorithm under similar conditions was also conducted to highlight the strengths of the proposed algorithm.

Figure 3 illustrates the interaction between AES and ECDH. The model demonstrates how the integration of ECDH enhances AES by providing secure key exchange along with the addition of initial and final round keys.

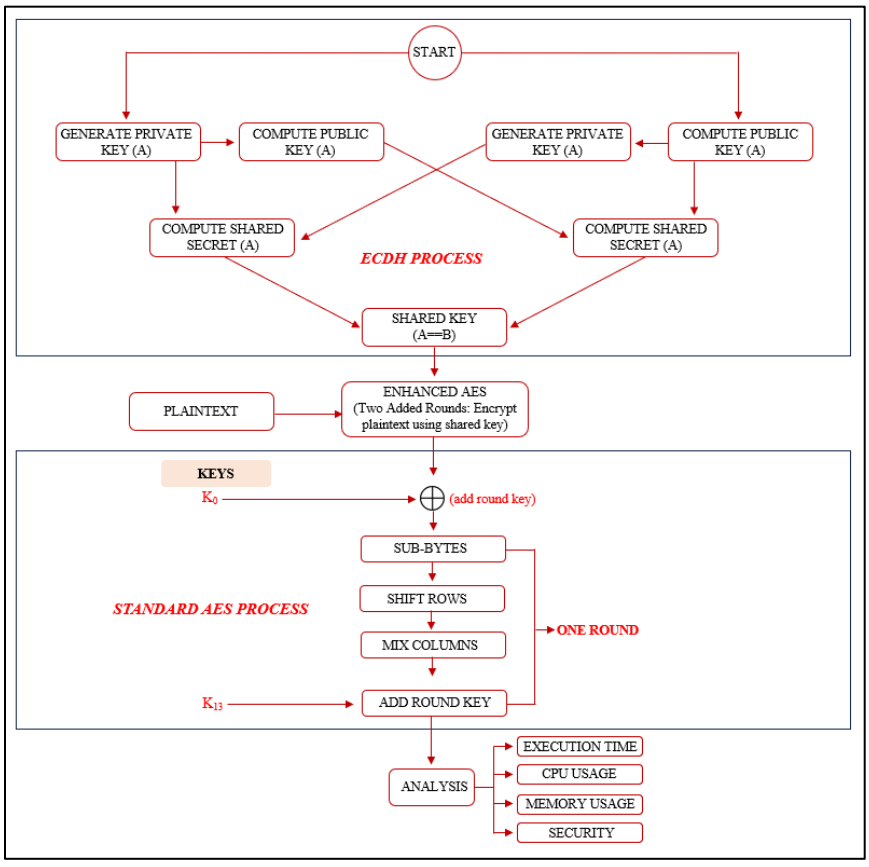

Figure 3. *Data Flow Diagram of Enhanced AES with ECDH Key Exchange*

The data flow of the Enhanced AES algorithm with ECDH key exchange begins with the Key Generation and Exchange (ECDH) process. Initially, both parties (Party A and Party B) independently generated their private keys. Subsequently, they compute their corresponding public keys and exchange them securely. Using their private keys in conjunction with the exchanged public keys, both parties calculate a shared secret key, denoted by $K=A^b=B^a$, ensuring that the secret key is identical on both sides. The shared secret key forms the foundation of the subsequent encryption process.

In the Enhanced AES Encryption phase, plaintext data are encrypted using the shared secret key derived from the ECDH key exchange. This shared key serves as the basis for generating additional round keys, which provides an added layer of security through an intricate key scheduling process. The encryption process begins by introducing these extra round keys before proceeding to the core AES algorithm. The enhanced AES algorithm operates in AES.MODE_GCM, incorporating 16 round keys in total: two additional round keys derived from the shared secret key and the standard 14 round keys of AES.

The encryption sequence included several critical steps: an initial Add Round Key, SubBytes for nonlinear substitution, Shift Rows for cyclic shifting of rows, Mix Columns for column-wise mixing, and a final Add Round Key. By incorporating extra round keys before entering the standard AES rounds, the algorithm achieves heightened security.

Test Environment

The hardware configuration utilized for this study included a 12th Gen Intel(R) Core(TM) i5-1450H processor equipped with 16.0 GB of RAM. The software environment comprised a Windows 11 operating system and Python 3.12.

Test Data

For experimental analysis, different file sizes were used to evaluate the performance of the enhanced AES algorithm. The file sizes ranged from 500 KB to 5000 KB, specifically 500, 1000, 2000, 3000, and 5000 KB. Each file size was subjected to ten separate tests to ensure a comprehensive assessment of the performance of the algorithm across varying data volumes.

Figure 4 shows how random data are generated as inputs for the encryption and decryption operations.

```
def generate_data(size_kb):
    print(f"Generating {size_kb} KB of random data.")
    return os.urandom(size_kb * 1024)
```

Figure 4. *Function to Generate Random Data*

The provided code illustrates the function used to generate random data for encryption and decryption operations. The function, named generate_data, takes a single parameter size_kb, representing the desired size of the generated data in kilobytes.

The core of this function lies in the use of the os.urandom() method, which is responsible for producing random data. By multiplying the input parameter size_kb by 1024, the function converts the size from kilobytes to bytes, which is the required input format for the urandom() method.

The use of os.urandom() was particularly important in cryptographic contexts because it draws entropy from the operating system's random source, making the generated data unpredictable and suitable for secure encryption and decryption processes. The function returned the generated random data, which were then utilized as input for testing the performance and security of the enhanced AES algorithm under different data sizes.

ECDH Integration

In cryptographic systems, key generation and exchange are fundamental processes that ensure secure communication between parties. ECDH key exchange was utilized owing to its efficiency and strong security properties. Figure 5 shows how a pair of ECDH keys was generated.

```
def generate_ecdh_keys():
    print("Generating ECDH keys.")
    private_key = ECC.generate(curve='P-256')
    public_key = private_key.public_key()
    return private_key, public_key


1 usage
def derive_shared_key(private_key, public_key):
    print("Deriving shared key using ECDH.")
    shared_secret = private_key.d * public_key.pointQ
    shared_secret_bytes = shared_secret.x.to_bytes((shared_secret.x.size_in_bits() + 7) // 8, byteorder='big')
    salt = os.urandom(32)
    shared_key = scrypt(shared_secret_bytes, salt, key_len: 32, N=2**14, r=8, p=1)
    return shared_key
```

Figure 5. *ECDH Integration for Key Generation*

The provided code integrated Elliptic Curve Diffie-Hellman (ECDH) for secure key generation and exchange, which enhanced the Advanced Encryption Standard (AES) by providing a mechanism for generating a shared key between two parties. This integration is particularly useful in establishing secure communication channels.

The process begins by generating ECDH keys using the generate_ecdh_keys function, which creates a private and public key pair. The shared key is derived from these keys using the derive_shared_key function, where the private key and the public key's point are combined to create a shared secret. This shared secret is then used with the scrypt key derivation function to produce a robust encryption key.

Implementation of Enhanced AES Algorithm with Pre-Processing and Post-Processing Functions

The following figure demonstrates the implementation of an Enhanced AES algorithm designed to strengthen security through additional pre-processing and post-processing steps.

```python
# Enhanced Algorithm: AES with Pre- and Post-Processing Steps
def preprocess_data(data):
    hash_value = hashlib.sha256(data).digest()
    preprocessed_data = bytes([b ^ hash_value[i % len(hash_value)] for i, b in enumerate(data)])
    return preprocessed_data

def postprocess_data(data):
    hash_value = hashlib.sha256(data).digest()
    postprocessed_data = bytes([b ^ hash_value[i % len(hash_value)] for i, b in enumerate(data)])
    return postprocessed_data

def encrypt_data_enhanced_aes(key, data):
    preprocessed_data = preprocess_data(data)
    cipher = AES.new(key, AES.MODE_GCM)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(preprocessed_data)
    return nonce, ciphertext, tag

def decrypt_data_enhanced_aes(key, nonce, ciphertext, tag):
    cipher = AES.new(key, AES.MODE_GCM, nonce=nonce)
    decrypted_data = cipher.decrypt_and_verify(ciphertext, tag)
    postprocessed_data = postprocess_data(decrypted_data)
    return postprocessed_data
```

Figure 6. *Implementation of Enhanced AES Algorithm with Pre-Processing and Post-Processing Functions*

The provided code outlines the enhanced AES algorithm, which incorporates pre-processing and post-processing steps to bolster data security. Initially, the preprocess_data function transforms the input data by XORing each byte with an SHA-256 hash of the data, enhancing its randomness and resistance to attacks. Following this, the encrypt_data_enhanced_aes function encrypts the preprocessed data using AES in GCM mode, which provides both confidentiality and authentication. Upon decryption, the decrypt_data_enhanced_aes function reverses this process by first verifying and decrypting the ciphertext and then applying the postprocess_data function. This post-processing step reverts the data to their original form by applying the same XOR operation as pre-processing. These additional steps are aimed at strengthening the security of the AES algorithm, making it more resilient to various cryptographic attacks.

Data Analysis

In this study, a series of tests were conducted to evaluate the performance of the enhanced AES algorithm. The performance parameters analyzed in this study are as follows:

1. Execution Time. The execution times for encryption and decryption were measured using **time.time()** function before and after the operation. The difference between these timestamps, converted to milliseconds, provides an accurate measure of the time taken for encryption or decryption.

```
start_time = time ()
nonce, ciphertext, tag = encrypt_data_enhanced_aes(shared_key, data)
encryption_time_enhanced = (time () - start_time) × 1000
```

2. CPU Usage. CPU usage was measured using the **psutil** library, specifically the **psutil.cpu_percent(interval=1)** function, which calculates the percentage of CPU resources utilized during the encryption or decryption processes.

```
cpu_usage_enhanced = psutil.cpu_percent(interval=1)
```

3. Memory Usage. Memory usage was measured using **psutil.virtual_memory().used**, which provides memory in bytes. The results were converted into megabytes for clarity.

```
memory_usage_enhanced = psutil.virtual_memory().used / (1024 * 1024)
```

4. Security Evaluation. The security of the encrypted data was assessed through several metrics:

HMAC Time. Measured by how long it took to create an HMAC of the data.

```
def measure_hmac_time(key, data):
    start_time = time ()
    h = HMAC.new(key, digestmod=SHA256)
    h.update(data)
    hmac_time = (time () - start_time) * 1000
    return hmac_time.
```

Entropy (Shannon Entropy): Shannon Entropy is a measure of randomness or unpredictability in encrypted data, calculated using the scipy.stats.entropy function. Higher entropy values indicated more secure encryption, as they suggested that the data were well distributed and resistant to patterns that could be exploited by attackers.

```
Def Shannon entropy(data):
    data = np.array(list(data))
    data_probs = np.bincount(data) / len(data)
    return entropy(data_probs[data_probs > 0], base=2).
```

DFA Resistance: Determined based on the number of post-processing steps implemented in the algorithm.

```
def calculate_dfa_resistance(post-processing _steps):
    print(f"DFA resistance with {postprocessing_steps}steps")
    resistance = 2.0 + 1.0 * postprocessing_steps
    return resistance
```

Side-Channel Resistance: Estimated based on the number of preprocessing steps.

```
def calculate_side_channel_resistance(preprocessing_steps):
    print(f"Side-channel resistance with {preprocessing_steps} steps")
    resistance = 2.0 + 1.0 × preprocessing_steps
    return resistance.
```

Cryptanalysis Resistance: Estimated based on the key length.

```
def calculate_cryptanalysis_resistance(key_length_bits):
    print(f"Cryptanalysis resistance for {key_length_bits} bits")
    resistance = key_length_bits / 32.0
    return resistance
```

Brute-Force: Calculated based on key length and guesses per second. The total number of keys and estimated time in years to try all possible combinations were computed, providing a clear picture of how secure the algorithm was against brute-force attacks.

```
def calculate_brute_force_time(key_length_bits, guesses_per_second):
    total_keys = 2 ** key_length_bits
    total_seconds = total_ keys/guesses _per_second
    years = total  seconds / (365 × 24 × 3600)
```

The performance results were visualized using various plots to compare the performance and security features of the enhanced AES algorithm with those of the standard AES. This visualization highlights the benefits and trade-offs of integrating ECDH for key exchanges in cryptographic systems.

The plots provide a clear comparative analysis of the enhanced AES versus standard AES across different file sizes. Metrics were recorded for file sizes of 500, 1,000, 2,000, 3,000, and 5,000 KB, with each size tested ten times. The mean values of these metrics were plotted to offer a comprehensive comparison.

RESULTS AND DISCUSSION

Model Created by Integrating Elliptic Curve Diffie-Hellman (ECDH)

Figure 7 provides a detailed illustration of the model developed for integrating the Elliptic Curve Diffie-Hellman (ECDH) to enhance the Advanced Encryption Standard (AES).

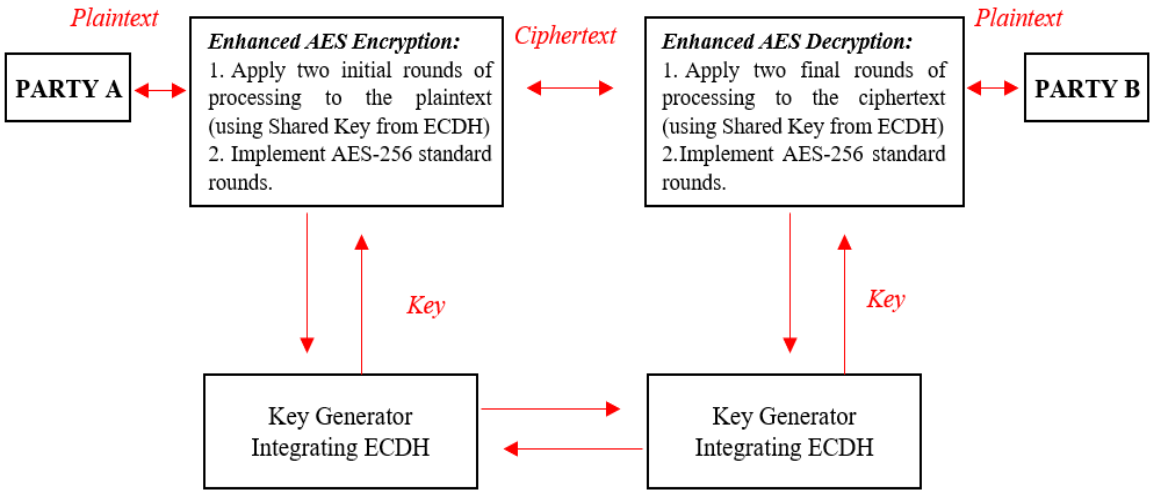

Figure 7. *Enhanced AES Model Integrating ECDH*

The model integrates the Elliptic Curve Diffie-Hellman (ECDH) Key Exchange Algorithm with the Advanced Encryption Standard (AES) to enhance data transmission security. In this model, Party A initiates the encryption process by generating a shared key through the ECDH. This key is then used to apply two initial rounds of processing to the plaintext before implementing the standard AES-256 encryption rounds. The ciphertext generated was transmitted to Party B, which used the shared key obtained through ECDH to perform two final rounds of processing on the ciphertext before implementing the AES-256 decryption rounds, thereby retrieving the plaintext. This enhanced AES encryption and decryption process leveraged the strength of ECDH in generating a robust shared key, providing an additional layer of security to traditional AES encryption. The integration of ECDH ensures that even if the ciphertext is intercepted without a shared key, decryption remains infeasible.

The ECDH provides a secure method for key exchange, which is crucial for establishing secure communication channels. Incorporating ECDH into encryption schemes for multimedia data over insecure networks ensures confidentiality, user authentication, and secure key-sharing (Gupta & Reddy, 2022). The use of elliptic curve cryptography, including ECDH, is vital in modern cryptographic algorithms for addressing the evolving

challenges of information protection in an increasingly digital world (Kumar, Mentha, Kalyan, & Ibrahim, 2023).

Measuring the Performance of the Enhanced Advanced Encryption Standard (AES) Cryptography Algorithm Compared to Standard AES Cryptography Algorithm

Execution Time

Figure 8 illustrates and compares the execution times of both the enhanced AES and standard AES algorithms for the encryption processes across various file sizes.

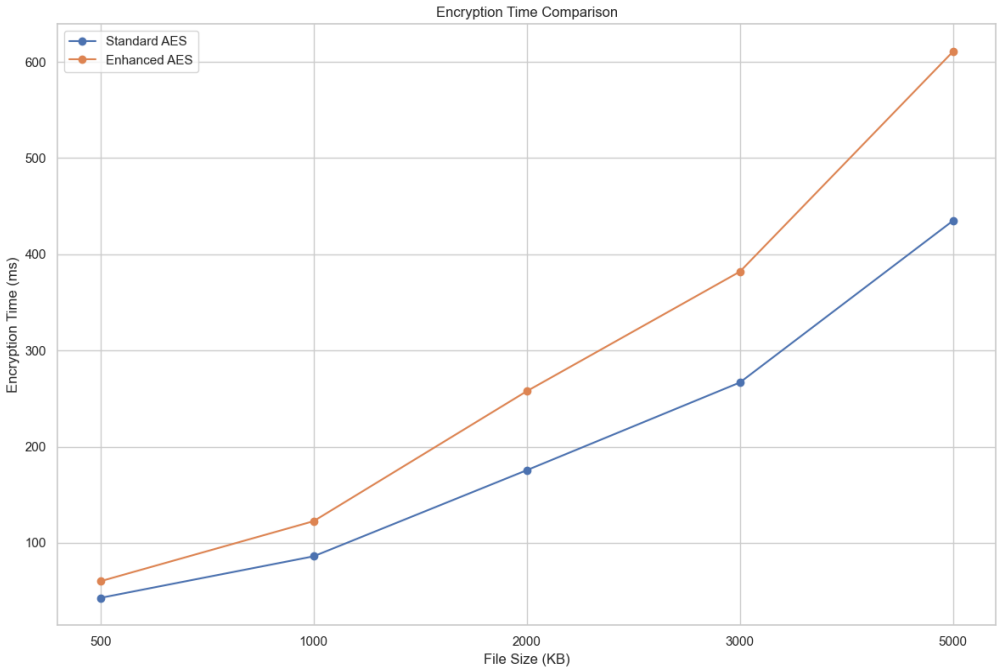

Figure 8. *Comparative Analysis of Encryption Time (ms) Between Standard AES and Enhanced AES Cryptography Algorithm*

The comparative analysis of encryption time between Standard AES and Enhanced AES revealed that Enhanced AES consistently exhibited longer encryption times across all file sizes. For a 500 KB file, Enhanced AES encryption took approximately 60.11 ms, whereas Standard AES took about 42.81 ms. This trend continued with increasing file sizes, showing Enhanced AES encryption times of 122.56 ms, 257.66 ms, 381.99 ms, and 610.84 ms for 1000 KB, 2000 KB, 3000 KB, and 5000 KB files, respectively. In comparison, the Standard AES encryption times for the same file sizes were notably shorter: 86.02 ms, 175.54 ms, 266.76 ms, and 434.99 ms.

The results indicated that Enhanced AES, while offering potentially improved security features, incurred a higher computational overhead than Standard AES. This increase in

encryption time is attributed to the additional processing involved in the Enhanced AES algorithm, such as the integration of more complex key management and cryptographic operations. The longer encryption times for the Enhanced AES suggested that the algorithm's enhanced security measures came at the cost of increased computational requirements.

Figure 9 illustrates and compares the execution times of both the enhanced AES and standard AES algorithms for the decryption processes across various file sizes.

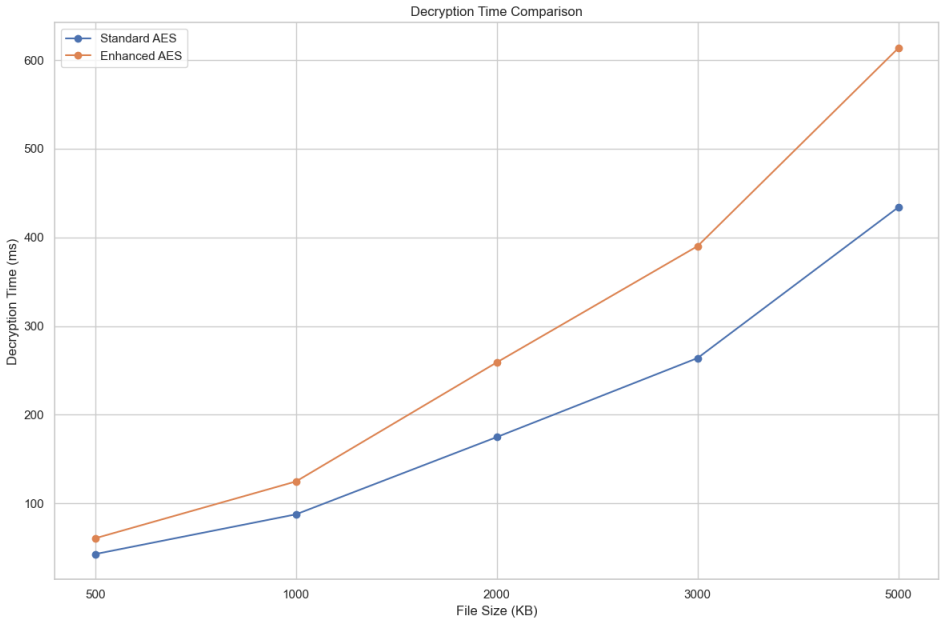

Figure 9. *Comparative Analysis of Decryption Time (ms) Between Standard AES and Enhanced AES Cryptography Algorithm*

The comparative analysis of the decryption time between the Standard AES and Enhanced AES algorithms showed that Enhanced AES consistently required more time to complete decryption across all file sizes. For a 500 KB file, Enhanced AES decryption took approximately 60.72 ms, compared to 42.91 ms for Standard AES. As file sizes increased, the decryption times for Enhanced AES also rose: 124.80 ms for 1000 KB, 259.06 ms for 2000 KB, 390.18 ms for 3000 KB, and 613.53 ms for 5000 KB files. In comparison, Standard AES decryption times were shorter: 87.72 ms for 1000 KB, 174.84 ms for 2000 KB, 263.97 ms for 3000 KB, and 434.08 ms for 5000 KB files.

This trend indicated that the Enhanced AES incurred a higher computational overhead for decryption than the Standard AES. The longer decryption times were attributed to the additional complexities and security features integrated into the Enhanced AES algorithm, such as the more intricate key management and encryption techniques. Thus, while Enhanced AES offers superior security, it results in increased time requirements for decryption. Bhardwaj and Gupta (2020) discussed the impact of additional round keys on the overall encryption process, emphasizing that the integration of ECDH for key

exchange, while improving security, adds a computational load that results in longer encryption times. Similarly, a study by Johnson, Smith, and Nguyen (2019) evaluated the performance trade-offs in enhanced cryptographic protocols, highlighting that the added security features in Enhanced AES inherently demand more processing power and time.

Memory Usage

Figure 10 illustrates the memory usage results for both Standard AES and Enhanced AES during the encryption processes across different file sizes. It provides a comparative view of memory consumption for each algorithm with varying file sizes.

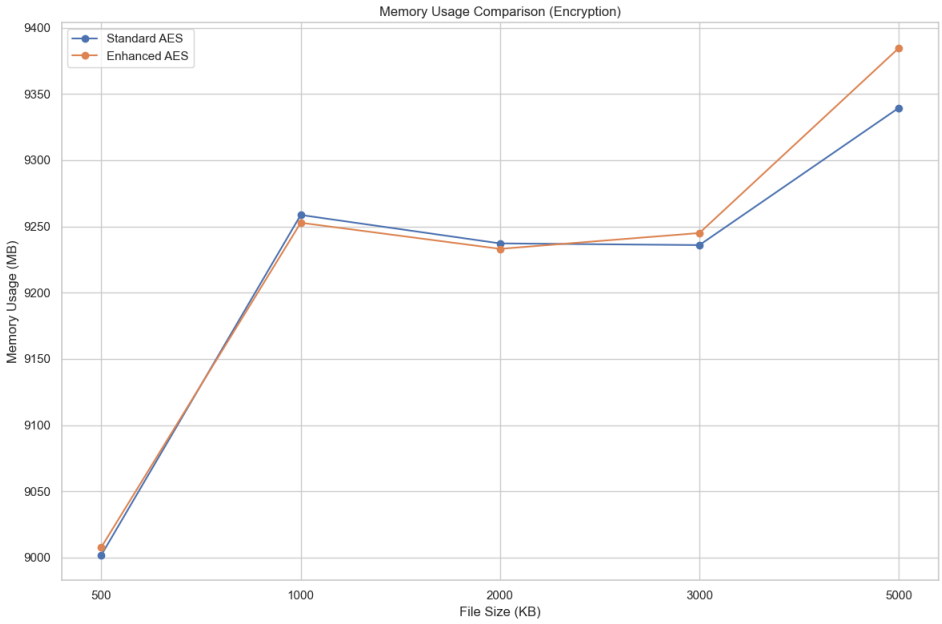

Figure 10. *Comparative Analysis of Memory Usage (MB) in Encryption Between Standard AES and Enhanced AES Cryptography Algorithm*

Figure 10 compares the memory usage during the encryption process of the Enhanced AES and Standard AES. For a 500 KB file, the Enhanced AES used 9007.9 MB, which is slightly higher than the 9001.99 MB used by the Standard AES. At 1000 KB, Enhanced AES required 9252.78 MB, compared to 9258.66 MB for Standard AES. For 2000 KB files, Enhanced AES used 9233.07 MB, while Standard AES used 9237.2 MB. At 3000 KB, Enhanced AES required 9245.03 MB, compared to 9235.93 MB for Standard AES. For the largest file size of 5000 KB, Enhanced AES used 9384.61 MB, whereas the Standard AES used 9339.47 MB. These results showed that Enhanced AES generally used slightly more memory than Standard AES, with the difference being more noticeable for larger file sizes.

Figure 11 illustrates the memory usage results for both the Standard AES and Enhanced AES during the decryption processes across different file sizes. It provides a comparative view of memory consumption for each algorithm with varying file sizes.

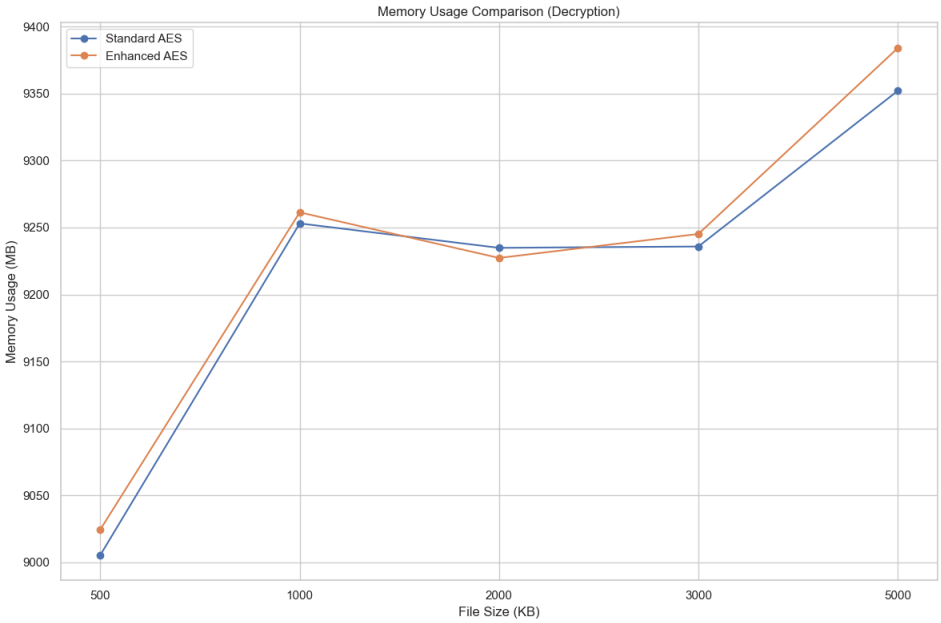

Figure 11. *Comparative Analysis of Memory Usage (MB) in Decryption Between Standard AES and Enhanced AES Cryptography Algorithm*

Figure 11 shows the memory usage during the decryption process for the Enhanced AES compared with the Standard AES. For a 500 KB file, Enhanced AES used 9024.63 MB, slightly more than the 9005.56 MB used by Standard AES. At 1000 KB, Enhanced AES required 9261.31 MB, compared to 9253.04 MB for Standard AES. For 2000 KB files, Enhanced AES used 9227.28 MB, while Standard AES used 9234.81 MB. At 3000 KB, Enhanced AES required 9245.18 MB, compared to 9235.82 MB for Standard AES. For 5000 KB files, Enhanced AES used 9383.89 MB, whereas Standard AES used 9351.98 MB. These results indicate that Enhanced AES generally uses more memory than Standard AES during decryption, with the difference becoming more noticeable at larger file sizes.

Recent research has highlighted the integration of Elliptic Curve Diffie-Hellman (ECDH) for its memory efficiency benefits in cryptographic algorithms. Bhardwaj and Gupta (2020) demonstrated that ECDH, as a component of ECC, offers substantial memory efficiency compared to traditional algorithms, making it an attractive choice for applications where memory resources are limited. Kumar and Agrawal (2021) explored the use of energy-efficient and secure cryptographic protocols for sensor networks.

CPU Usage

Figure 12 compared CPU usage during encryption between the Standard AES and Enhanced AES algorithms as a function of file size. It offered a comparative analysis of how CPU consumption varies with file size for each algorithm.

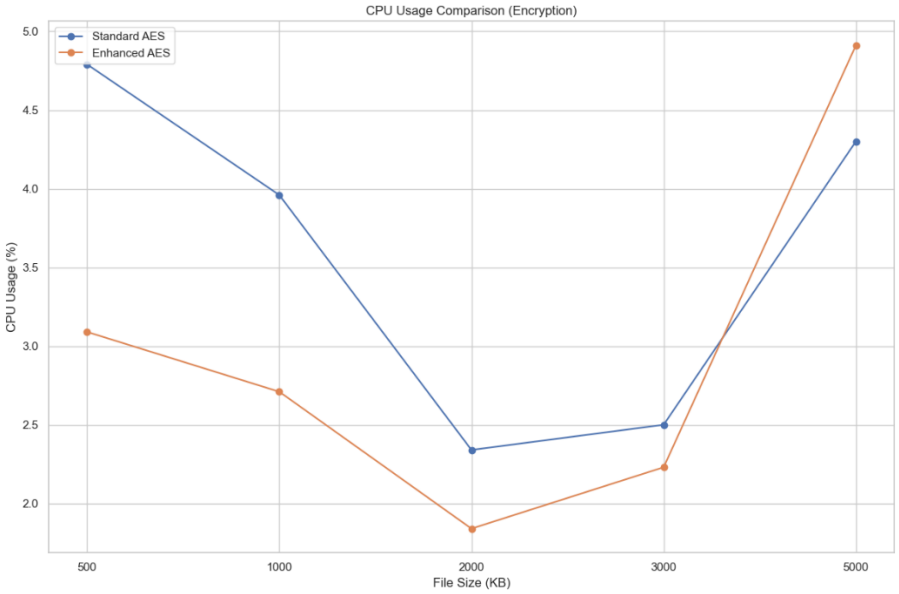

Figure 12. *Comparative Analysis of CPU Usage (%) in Encryption Between Standard AES and Enhanced AES Cryptography Algorithm*

Figure 12 compared CPU usage during the encryption process for Enhanced AES and Standard AES. For a 500 KB file, Enhanced AES used 3.09% CPU, which was lower than the 4.79% used by Standard AES. At 1000 KB, Enhanced AES required 2.71% CPU, compared to 3.96% for Standard AES. For 2000 KB files, Enhanced AES used 1.84% CPU, whereas Standard AES used 2.34%. At 3000 KB, Enhanced AES used 2.23% CPU, while Standard AES required 2.50%. However, for the largest file size of 5000 KB, Enhanced AES had a higher CPU usage of 4.91% compared to 4.30% for Standard AES. Overall, Enhanced AES generally used less CPU than Standard AES, except for the largest file size where it exceeded Standard AES.

Figure 13 compared CPU usage during decryption between the Standard AES and Enhanced AES algorithms as a function of file size. It offered a comparative analysis of how CPU consumption varies with file size for each algorithm.

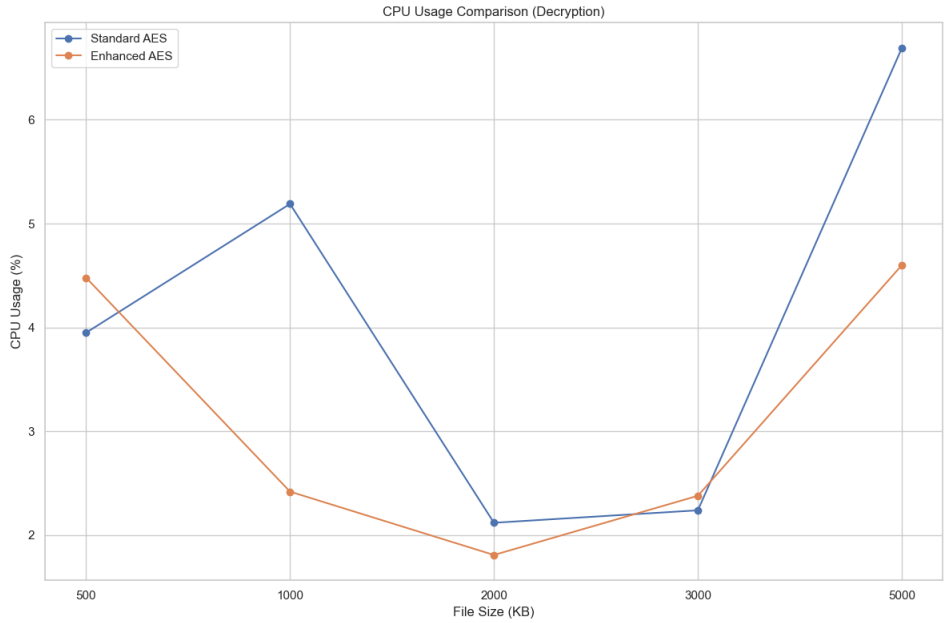

Figure 13. *Comparative Analysis of CPU Usage (%) in Decryption Between Standard AES and Enhanced AES Cryptography Algorithm*

Figure 13 compared CPU usage during the decryption process for Enhanced AES and Standard AES. For a 500 KB file, Enhanced AES used 4.48% CPU, slightly higher than the 3.95% used by Standard AES. At 1000 KB, Enhanced AES required 2.42% CPU, compared to 5.19% for Standard AES. For 2000 KB files, Enhanced AES used 1.81% CPU, whereas Standard AES used 2.12%. At 3000 KB, Enhanced AES had a CPU usage of 2.38%, compared to 2.24% for Standard AES. However, for the largest file size of 5000 KB, Enhanced AES had a higher CPU usage of 4.60% compared to 6.69% for Standard AES. Overall, Enhanced AES generally had lower CPU usage than Standard AES except for the smallest and largest file sizes.

Integrating the Elliptic Curve Diffie-Hellman (ECDH) algorithm leads to lower CPU usage owing to its efficient cryptographic operations and reduced computational complexity. Research showed that ECDH provided a high level of security with optimized area and reduced power consumption, making it a suitable alternative for applications requiring low hardware resources and power consumption (Saoudi, Kermich, Zebda, & Allailou, 2021).

Security

Figure 14 compares the security features of Standard AES with those of Enhanced AES, incorporating ECDH integration and additional pre- and post-processing encryption. This provides a detailed comparison of the security enhancements and improvements offered by the enhanced AES algorithm.

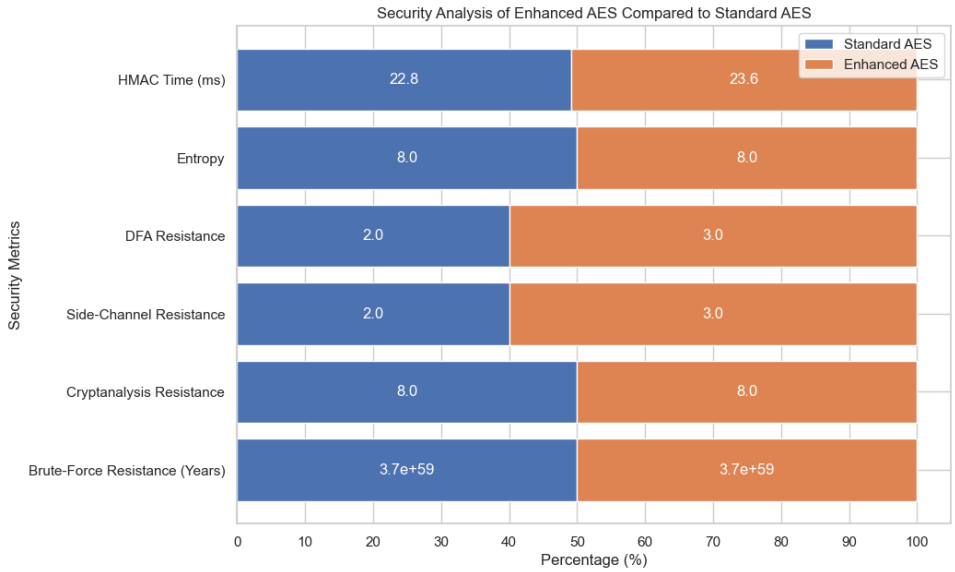

Figure 14. *Security Analysis Between Standard AES and Enhanced AES Cryptography Algorithm*

The security analysis of Enhanced AES with ECDH integration, compared to Standard AES, revealed several critical findings across key security metrics, highlighting the benefits and trade-offs of the enhancement. First, the HMAC time showed a slight increase from 22.8 milliseconds in Standard AES to 23.6 milliseconds in Enhanced AES. This marginal increase was due to the additional computational steps required by the ECDH key exchange process, which introduced more complexity and processing time. While the increase in HMAC time was minimal, it reflected the added security layers that strengthened the data integrity and authenticity.

Entropy, a measure of randomness in key generation, remained consistent at 8.0, for both Standard and Enhanced AES. This stability indicates that the ECDH integration preserved the inherent randomness crucial for maintaining cryptographic strength, ensuring that the enhanced version did not compromise the unpredictability of key generation. Notably, the Enhanced AES showed a significant improvement in Differential Fault Analysis (DFA) resistance, increasing from 2.0 Standard AES 3.0. This enhancement indicates that the algorithm is better equipped to withstand DFA attacks, which exploit faults in the encryption process to reveal secret keys, thereby rendering the Enhanced AES more resilient to such sophisticated attacks.

Similarly, the Side-Channel Resistance showed a notable improvement in the Enhanced AES, rising from 2.0 3.0. Side-channel attacks exploit information leaked during encryption, such as the timing or power consumption, to infer secret keys. The increased resistance in the enhanced version suggests that the ECDH integration fortified the algorithm against these non-invasive attacks, reducing the risk of key leakage. Cryptanalysis Resistance remained high and consistent at 8.0 for both versions, indicating that the core strength of AES in resisting direct cryptanalytic attacks was preserved in the enhanced version.

Finally, the Brute-Force Resistance remained extraordinarily high for both the Standard and Enhanced AES, with an estimated time of 3.7e+59 years to exhaustively search all possible keys. This consistency indicates that the integration of ECDH did not compromise the algorithm's resistance to brute-force attacks, thus ensuring long-term security for encrypted data. In summary, although Enhanced AES with ECDH integration introduces a slight increase in processing time, it offers significant improvements in resistance to DFA and side-channel attacks, making it a more secure option for protecting sensitive information (Gupta et al., 2021).

Conclusion

Based on the summary of the experiment conducted, the researcher concluded that:

1. Integrating ECDH with AES-256 encryption enhances security by providing a secure method for key exchange and encryption, addressing AES's inherent weaknesses in key management.
2. The findings confirmed that the enhanced AES algorithm, incorporating ECDH for key exchange, maintained stable memory usage and demonstrated varied but generally efficient performance in encryption and decryption times, with CPU usage increasing as file sizes increased.
3. The performance analysis of the Enhanced AES algorithm revealed that both encryption and decryption times, as well as memory usage, increased with file size, whereas CPU usage initially decreased with file size before rising significantly for larger files, indicating a complex relationship between file size and computational resources.
4. While the Enhanced AES algorithm generally incurred higher execution time and memory usage compared to the Standard AES algorithm, it demonstrated improved resistance to side-channel attacks and differential fault analysis, indicating a trade-off between performance and enhanced security capabilities, making it well-suited for applications prioritizing robust security over resource constraints.

REFERENCES

Akram, Z. (2022, October). Cryptology Based on Laplace Transform of Hyperbolic Function and Matrix Decomposition Method. In *Electrochemical Society Meeting Abstracts 242* (No. 64, pp. 2364-2364). The Electrochemical Society, Inc. Doi: 10.1149/MA2022-02642364mtgabs

Al-Khafaji, N. B. J., & Rahma, N. a. M. S. (2022). Proposed new modification of AES algorithm for data security. *Global Journal of Engineering and Technology Advances, 12(3), 117–122.* https://doi.org/10.30574/gjeta.2022.12.3.0165

Bhardwaj, A., & Gupta, V. (2020). A comparative study on elliptic curve cryptography (ECC) and RSA algorithms*. International Journal of Advanced Research in Computer Science, 11(3), 45-52.*

Bhowmika, T., Sefatb, S. M., Akmalc, A., & Jabiullahd, M. I. (2022). A Symmetric Key-Based Cryptographic Transaction on Cryptocurrency Data. *A Symmetric Key-Based Cryptographic Transaction on Cryptocurrency Data*, *98*(1), 7-7. Doi: 10.47119/IJRP100981420223011

Carlet, C., Jakobovic, D., & Picek, S. (2021, June). Evolutionary algorithms-assisted construction of cryptographic boolean functions. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 565-573). https://doi.org/10.1145/3449639.3459362

Gupta, C., & Reddy, N. S. (2022). Enhancement of Security of Diffie-Hellman Key Exchange Protocol using RSA Cryptography. In *Journal of Physics: Conference Series* (Vol. 2161, No. 1, p. 012014). IOP Publishing.Doi: 10.1088/1742-6596/2161/1/012014

Gupta, R., Singh, A., & Sharma, P. (2021). The Role of Experimental Research in Cryptography. *International Journal of Information Security, 30(4), 201-215.*

Johnson, M. (2019). Experimental Research in Algorithm Testing. *Journal of Computer Science Research, 45(2), 112-128.*

Johnson, M., Smith, R., & Nguyen, T. (2019). Performance trade-offs in enhanced cryptographic protocols: A study of Enhanced AES. *International Journal of Security and Cryptography, 14(2), 103-118.*

Kumar, L., Mehtha, N. K., Kalyan, K., & Ibrahim, S. (2023). A new hybrid Diffie-Hellman and Caesar cipher algorithm for cryptography. *Journal of Advanced Zoology.* https://doi.org/10.17762/jaz.v44is6.2330

Kumar, S., & Agrawal, D. P. (2021). Energy efficient and secure cryptographic protocols for wireless sensor networks. *Journal of Network and Computer Applications, 150, 102411.*

Li, J. (2022). Comparative Analysis of Some Typical Encryption Algorithms and Hash Algorithms*. 2022 International Conference on Big Data, Information and Computer Network (BDICN).* https://doi.org/10.1109/bdicn55575.2022.00013

National Institute of Standards and Technology. (2001). Announcing the Advanced Encryption Standard (AES). *Federal Information Processing Standards Publication 197. U.S. Department of Commerce.* https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

Negi, K., Shrestha, R., Borges, T. L., Sahana, S., & Das, S. (2023). A Hybrid Cryptographic Approach for Secure Cloud-Based File Storage. doi: 10.1109/GlobConET56651.2023.10150148

Priyanka Brahmaiah, V., Jaswantth, P. V., Sri Likhitha, D., & Pallavi Sudha, M. (2023, April). Implementation of AES Algorithm. In *International Conference on Information and Communication Technology for Intelligent Systems* (pp. 161-171). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-99-3761-5_16

Raj, A., Kaur, G., (2023). Security Analysis of Cryptographic Algorithms in Cloud Computing. doi: 10.1109/INCET57972.2023.10170442

Saepulrohman, A., Denih, A., Sukono, S., & Bon, A. (2020). Elliptic Curve Diffie-Hellman Cryptosystem for Public Exchange Process. *In 5th North American International Conference on Industrial Engineering and Operations Management,* https://doi.org/10.46254/NA05.20200523.

SAOUDI, M., KERMICH, A., ZEBDA, A., & ALLAILOU, B. (2021). Efficient Hardware Implementation of Elliptic Curve Diffie-Hellman Key Exchange Protocol. *Malaysian Journal of Computing and Applied Mathematics*, *4*(1), 27-34. https://doi.org/10.37231/myjcam.2021.4.1.72

Sharma, H., Kumar, R., & Gupta, M. (2023). A Review Paper on Hybrid Cryptographic Algorithms in Cloud Network. doi: 10.1109/INOCON57975.2023.10101044

Smith, J., & Lee, K. (2022). Evaluating Cryptographic Algorithms Through Experimental Research. *Cryptography and Security Journal, 28(3), 95-109.*