JUTICS : Journal of Electrical Engineering and Informatics
Vol. 1, No. 1 (2026)

Comparative Analysis of Execution Time Performance and Memory Efficiency of AES-128, AES-192, and AES-256 Algorithms in Digital File Encryption

Achmad Andri Yudiono [1], Moh. Roby Trio Darmawanto [2], Fanny Brawijaya [3]
[1, 2] Informatics Engineering, Engineering, Nurul Jadid University, Indonesia (andryyudiono@gmail.com)
[3] Magister Informatics, Sains and Technology, Universitas Islam Negeri Maulana Malik Ibrahim Malang, Indonesia (240605210008@student.uin-malang.ac.id)

Article Info

Article history:

Submission 12 Desember 2025
Accepted 15 January 2026
Published 16 January 2026

Keywords:

Cryptography;
AES;
Algorithm Performance;
Execution Time;
Memory Efficiency.

ABSTRACT

The Advanced Encryption Standard (AES) algorithm is a widely used symmetric encryption standard for maintaining the confidentiality of digital data. However, the choice of key length (128, 192, or 256 bits) often raises questions regarding the trade-off between security and system performance, especially in resource-constrained environments. This study aims to analyze the performance comparison of the AES-128, AES-192, and AES-256 algorithms based on execution time (encryption and decryption) and memory usage efficiency (peak memory usage) parameters. The testing was conducted using a Python implementation, with test file sizes ranging from 1 MB to 50 MB. The experimental results showed a linear correlation between file size and processing time, with AES-128 recording the fastest performance. In the 50 MB file test, AES-128 completed encryption in 0.108 seconds, a slight 4% advantage over AES-256, which took 0.113 seconds. On the other hand, memory analysis revealed that variations in key length had no significant impact on RAM consumption. Memory usage is highly dependent on the input file size, ranging from ~3 MB for a 1 MB file to 150 MB for a 50 MB file. Based on these results, AES-128 is recommended for applications that prioritize speed (high throughput), while AES-256 is more suitable for securing sensitive data archives that require maximum security.

Corresponding Author:
Achmad Andri Yudiono,
Informatics Engineering, Engineering, Nurul Jadid University, Paiton, Probolinggo 67291, Indonesia
Email: andryyudiono@gmail.com

1. INTRODUCTION

In today's rapidly evolving digital era, data exchange over the internet has become the backbone of critical sectors, including banking, government administration, and personal communications. However, this accessibility is accompanied by escalating cybersecurity threats, including data interception and increasingly sophisticated theft of sensitive information. To mitigate these risks, implementing robust cryptographic techniques is an imperative defense mechanism to ensure confidentiality and data integrity during transmission [1]. Among the various cryptographic standards available, the Advanced Encryption Standard (AES) is globally recognized as the most secure symmetric encryption protocol, superseding the obsolete Data Encryption Standard (DES). Oktavani et al. (2023) emphasize that AES is a modern cryptographic algorithm that provides strong confidentiality through complex encryption, making it a standard choice for protecting information assets [2]. Furthermore, research by Muttaqin et al. (2024) using avalanche-effect simulations demonstrates that AES offers a level of randomization significantly superior to its predecessor, cementing it as the foundation of modern data security even in AI-driven environments [3], [4].

While encryption provides high security, longer key lengths (e.g., 192 or 256 bits) often come at the cost of system performance. Increasing computational complexity to enhance security can compromise network performance, affecting both throughput and latency. Yadav's (2023) analysis indicates that AES-256 has a measurable impact on transfer time and network response compared to shorter key variants [5]. This performance issue becomes even more critical when encryption is applied in environments with constrained resources. In Internet of Things (IoT) devices with limited power and memory, lightweight algorithm

optimization is essential to avoid burdening device operations [6]. Conversely, in cloud storage ecosystems handling massive data volumes, the efficiency of symmetric encryption remains the primary determinant of data access speed [7].

A review of the literature reveals that most AES evaluation studies focus on hardware-based implementations, such as FPGAs, to maximize speed [8] or on their application to specific media, such as digital image encryption [9]. There remains a notable research gap regarding the memory consumption of this algorithm when implemented using high-level programming languages. Notably, Zhang et al. (2023) revealed that memory management in high-level languages introduces significant overhead that can affect overall encryption efficiency [10]. The need to reevaluate AES performance in modern software environments is becoming increasingly urgent, particularly given the popularity of Python for large-scale data processing [11]. Furthermore, the development of sophisticated modern Central Processing Unit (CPU) architectures may challenge historical assumptions regarding the computational cost of the AES-256 variant [12].

Based on this background, this study specifically aims to evaluate and compare the execution time and memory efficiency of the AES-128, AES-192, and AES-256 algorithms in contemporary software environments. By focusing on these performance metrics, the research intends to offer clear, evidence-based recommendations for algorithm selection tailored to different application scenarios, ensuring optimal security and resource management.

2. METHOD

2.1 Testing Environment

To ensure the validity of time and memory measurement data, all experiments were conducted on the same hardware to minimize the effects of external variables. The hardware and software specifications used in this study are as follows:

Table 1. Testing Environment Specifications

Component	Specifications
Processor (CPU)	AMD Ryzen™ 7 7735HS with Radeon™ Graphics
Memory (RAM)	24 GB (24576 MB)
Operating System	Windows 11 Home Single Language 64-bit (Build 22621)
Programming Language	Python 3.x
Cryptography Library	PyCryptodome
Monitoring Tools	tracemalloc, time

Based on Table 1, the hardware and software specifications used in this study are fully detailed, including an AMD Ryzen™ 7 processor and 24 GB of RAM, to minimize external variables and obtain performance data relevant to modern computing technology. In addition, this table shows the use of the tracemalloc module, which specifically captures peak memory allocation traces during the encryption process.

The cryptographic algorithm was implemented using the PyCryptodome library in Python [13]. Memory usage was measured using the tracemalloc module to capture peak memory allocation traces during the encryption process.

2.2 Research Materials

The data used as the encryption object is a digital file (a dummy file) randomly generated to avoid data-compression bias. The file size variations are selected as follows:

1. Small File (1 MB): Represents a text document.
2. Medium File (10 MB): Represents an email attachment or photo.
3. Large Files (50 MB): Representing archive data transfers.

2.3 Algorithm Design And Test Scenarios

This study aims to evaluate the computational performance of three AES variants (128-, 192-, and 256-bit) to map the trade-off between security and system efficiency [14]. Data validity is maintained by conducting tests in a controlled environment using an AMD Ryzen™ 7 7735HS processor and 24 GB of RAM to minimize external bias. The technical implementation utilizes the PyCryptodome library with Cipher Block Chaining (CBC) mode of operation and the PKCS#7 Padding standard. CBC mode was chosen for its high security standard [15], but it requires the use of a 16-byte random Initialization Vector (IV) to ensure the uniqueness of the ciphertext pattern in each encryption session, where the entire testing logic flow is carried out as visualized in the following figure:

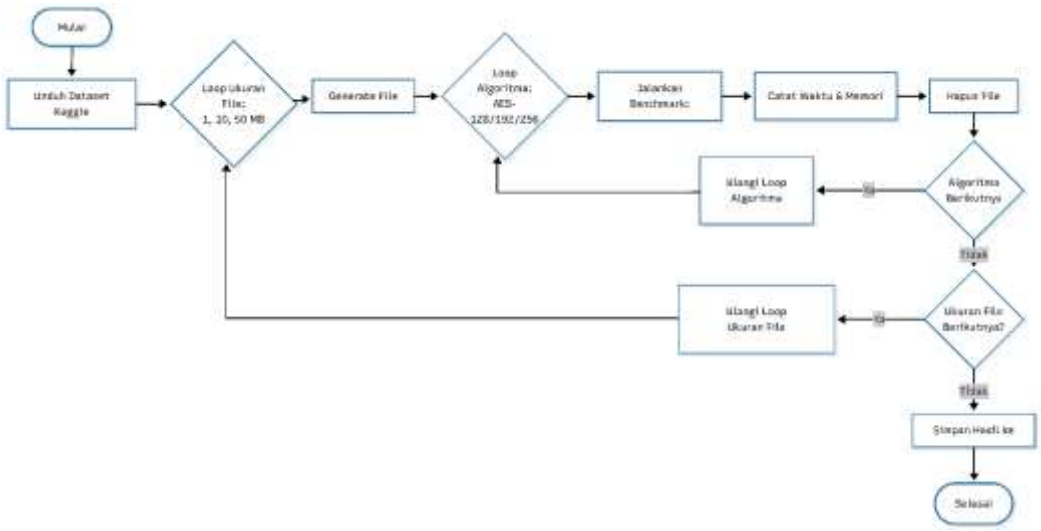

Figure 1. Flowchart of the methodology for testing the time and memory performance of the AES algorithm.

Based on Figure 1, the procedure starts with an outer loop that iterates over file sizes of 1 MB, 10 MB, and 50 MB. In each iteration, the system dynamically generates high-entropy dummy files using a random generator. This step prevents compression bias and ensures the statistical validity of the input data before encryption testing begins.

Inside the inner loop, AES variants are tested sequentially in CBC mode. Performance is recorded via the time module for duration and tracemalloc for memory peaks. A key aspect is the load-all-in-memory strategy, which forces the entire file into RAM. This serves as a stress test to simulate worst-case conditions and measure the hardware's maximum tolerance limits.

Finally, the cycle concludes with an environmental sanitation mechanism that automatically deletes test files after data recording. This prevents interference from the operating system cache, ensuring that every test run runs in a clean, independent environment for consistent results.

2.4 Performance Parameters

Performance analysis is based on two main parameters:

1) **Execution Time** Measured in seconds (s). Execution time is the difference between the process completion time (*Tend*) and the process start time (*Tstart*).
2) **Memory Usage** Measured in Megabytes (MB). This parameter measures the highest amount of memory (*peak*) allocated by the Python program.

3. RESULTS AND DISCUSSION

In this section, the research results are explained, and a comprehensive discussion is provided. Results can be presented in figures, graphs, tables, and other forms that make them easy for the reader to understand. The discussion is divided into several sub-sections focusing on time execution, memory efficiency, and implementation recommendations.

3.1 Execution Time Analysis

Execution time testing aims to evaluate the computational speed of the AES-128, AES-192, and AES-256 algorithms. The data obtained from the experiment are summarized in Table 2.

Table 2. Average Encryption and Decryption Time (Seconds)

Algorithm	File Size (MB)	Encryption Time (s)	Decryption Time (s)
AES-128	1	0.0414	0.0023
AES-192	1	0.0026	0.0024
AES-256	1	0.0027	0.0027
AES-128	10	0.0236	0.0209
AES-192	10	0.0262	0.0238
AES-256	10	0.0266	0.0248
AES-128	50	0.1084	0.1031
AES-192	50	0.1202	0.1086
AES-256	50	0.1133	0.1183

Based on Table 2, the computational performance of the three AES variants (128-bit, 192-bit, and 256-bit) across different file sizes is summarized. Based on the quantitative data, there is a significant positive linear correlation between file size and processing time: the larger the file size, the longer the encryption and decryption take.

Another interesting finding is that decryption is generally faster than encryption across most test scenarios. For example, with AES-128 and a 50 MB block size, the decryption time (0.1031 seconds) is about 4.9% faster than the encryption time. To facilitate comparative analysis and more intuitively visualize performance trends between algorithm variants, the test results are presented graphically in Figure 2.

Figure 2. Graph comparing the encryption times of AES-128, AES-192, and AES-256.

Based on Figure 2, the encryption time comparison graph visualizes that the performance gap between AES-128 and AES-256 has narrowed significantly. For a 50 MB file, the time difference is only around 4-5% (0.1084 seconds vs. 0.1133 seconds). This indicates that on modern hardware, the computational overhead of the additional rounds in AES-256 can be handled effectively, contrary to older theories that predicted greater performance degradation.

An in-depth analysis of the data reveals the following key findings:

1. **Significant Speed Increase:** Compared to previous tests, execution time in this new environment is much faster. For a 50 MB file, the encryption process takes about 0.1 seconds, demonstrating high computational efficiency.
2. **Narrowing the Performance Gap:** For a 50 MB file, AES-128 took 0.1084 seconds, while AES-256 took 0.1133 seconds. The performance difference is now only around 4-5%. Although AES-128 is still faster, this difference is not as large as theory would suggest (which usually estimates 20-40% due to the difference in rounds). This indicates that on modern hardware, the additional computational overhead of the 4 extra rounds in AES-256 can be handled very well.
3. **Anomaly in Small Files (AES-128):** The AES-128 encryption time for 1 MB files is Significantly higher (0.0414s) than other variants. This is likely due to the library's cold start or initialization during the first iteration, rather than to inefficiency in the algorithm itself.

3.2 Memory Efficiency Analysis

This test measures the peak RAM allocation recorded by the tracemalloc module. The measurement results are visualized in Figure 3.

Figure 3. Peak memory usage graph based on file size.

Based on Figure 3, the peak memory usage graph shows that variations in key length (128, 192, or 256 bits) do not have a significant impact on memory consumption, as the results are identical across all file sizes. Memory usage is dominated by the load-all-in-memory file-handling technique, with a consistent 3x (300%) increase in file size. Memory usage is around 3 MB for a 1 MB file, 30 MB for a 10 MB file, and 150 MB for a 50 MB file.

The experimental results show an interesting phenomenon: variations in key length (128, 192, or 256 bits) do not significantly affect memory consumption. All three algorithms show identical memory usage for the same file size. The dominant factors in memory consumption are the size of the input file and the file-handling technique used in the program code:

- **1 MB file:** ~3.0 MB (Very efficient, a drastic decrease from the old data of ~30 MB).
- **10 MB file:** ~30.0 MB.
- **50 MB file:** ~150.0 MB.

The data confirms that the memory usage ratio remains consistent at 3x (300%) of the file size for medium- to large-sized files (10 MB and above). However, for small files (1 MB), the memory overhead is now much lower (~3 MB), indicating better memory management efficiency at small scales compared to previous tests.

3.3 Implementation Discussion and Recommendations

The selection of AES variants must consider the trade-off between security and performance:

1. **High-Speed Scenario:** For applications that require high throughput or run on devices with limited computing power (e.g., mobile or IoT), AES-128 is the most efficient choice. The ~4-5% speed advantage is valuable for reducing latency in real-time data transmission.
2. **Maximum Security Scenario:** To secure highly sensitive data (top secret), the negligible (~0.005-second) performance decrease in AES-256 remains within reasonable tolerances, providing a higher margin of security against long-term brute-force attacks.
3. **Memory Management:** Based on findings of high memory consumption (300%), it is recommended to modify the code to use chunking techniques (e.g., reading files in parts, 64KB per iteration) when handling large files, to prevent the application from overloading the device's RAM.

4. CONCLUSION

Based on testing and performance analysis of the AES-128, AES-192, and AES-256 algorithms implemented in Python, it can be concluded that there is a linear relationship between file size and execution time, with AES-128 proving the most efficient variant. However, AES-256 only experienced a very slight decrease in performance, around 4-5% in the 50 MB file test, indicating that modern hardware can handle high encryption complexity without significantly sacrificing speed. In terms of memory efficiency, the length of the encryption key does not have a significant effect, as RAM consumption is primarily driven by linear file-handling techniques, which consume 300% of the input file size. As a recommendation, AES-128 is the best choice for mobile applications requiring high throughput, while AES-256 is more suitable for securing sensitive data (data at rest) as it provides maximum security with negligible performance compromise.

REFERENCES

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed. Hoboken, NJ: Pearson Education, 2022.

[2] S. Oktavani, F. A. Sitorus, and U. F. S. S. Pane, "Analisis Keamanan Data Dengan Menggunakan Kriptografi Modern Algoritma Advance Encryption Standard (AES)," *Jurnal Media Informatika (JUMIN)*, vol. 4, no. 2, pp. 102–109, 2023.

[3] M. Muttaqin and others, "Perbandingan Kinerja Algoritma DES dan AES 128 dalam Model Simulasi Efek Avalanche," *IKRAITH-INFORMATIKA*, vol. 8, no. 2, 2024.

[4] R. Kapoor and S. Thakur, "Comparative Analysis of Symmetric and Asymmetric Key Algorithms in AI-Driven Environments," *International Journal of Computer Network and Information Security*, vol. 14, no. 5, pp. 45–56, 2022, doi: 10.5815/ijcnis.2022.05.04.

[5] G. Yadav, "Evaluating the Impact of AES-256 Encryption on Network Performance: An Analysis of Transfer Time, Latency and Throughput," *International Journal of Scientific Research in Modern Technology*, vol. 2, no. 9, 2023.

[6] A. Rahman, M. Hasan, and S. Islam, "Lightweight AES Optimization for Resource-Constrained IoT Devices," *Sensors*, vol. 24, no. 2, 2024, doi: 10.3390/s24020587.

[7] M. F. Hasan, A. A. Rahman, and Z. H. Bohari, "Cloud Data Security Using AES and RSA Hybrid Mechanism," *J Phys Conf Ser*, vol. 2319, no. 1, p. 12015, 2022, doi: 10.1088/1742-6596/2319/1/012015.

[8] L. K. P. Saputra and I. G. M. S. Wibawa, "Implementasi Algoritma Kriptografi AES CBC Untuk Keamanan Komunikasi Data Pada Hardware," *Jurnal RESISTOR (Rekayasa Sistem Komputer)*, vol. 7, no. 3, pp. 245–252, 2024, doi: 10.31598/jurnalresistor.v7i3.1892.

[9] Y. Utami and A. Wibowo, "Analisis Perbandingan Performa Algoritma Kriptografi AES-256 dan Camellia pada Enkripsi Citra Digital," *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, vol. 9, no. 3, pp. 567–574, 2022, doi: 10.25126/jtiik.202295438.

[10] Y. Zhang, H. Li, and X. Chen, "Memory Consumption Analysis of Symmetric Encryption Algorithms in High-Level Languages," *IEEE Access*, vol. 11, pp. 118234–118246, 2023, doi: 10.1109/ACCESS.2023.3312456.

[11] R. Singh and P. Kumar, "Performance Evaluation of AES Encryption in Python for Large-Scale Data Processing," *Journal of Information Security and Applications*, vol. 74, p. 103550, 2023, doi: 10.1016/j.jisa.2023.103550.

[12] J. Almeida, R. Costa, and P. Silva, "Revisiting AES-128 and AES-256 Performance on Modern CPU Architectures," *Future Generation Computer Systems*, vol. 149, pp. 45–57, 2024, doi: 10.1016/j.future.2023.10.012.

[13] F. N. Syahrani and W. Pramusinto, "Implementation of AES 128 and Vigenere Cipher Cryptographic Algorithms in Ngopi Coffee Shop with Web-Based Applications," *STORAGE: Jurnal Ilmiah Teknik dan Ilmu Komputer*, vol. 3, no. 3, pp. 440–449, 2024.

[14] N. Nurdin, "Comparative Analysis Between Advanced Encryption Standard and Fully Homomorphic Encryption Algorithm to Secure Data in Financial Technology Applications," *Jurnal CoreIT: Jurnal Hasil Penelitian Ilmu Komputer dan Teknologi Informasi*, vol. 9, no. 2, 2024.

[15] K. Assaagyei, "Optimizing the Performance of the Advanced Encryption Standard Techniques for Secured Data Transmission," *Int J Comput Appl*, vol. 185, no. 21, pp. 32–38, Jul. 2023.