ISSN 2278-3091
Volume 14, No.3, May - June 2025
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse011432025.pdf
https://doi.org/10.30534/ijatcse/2025/011432025

Comparison of Cryptographic Performance in Python, Java, and C# Analyzing Language-Level Efficiency

Mildra March M. Tejano[1], **Ervin Simon D. Uadan**[2]
[1] North Eastern Mindanao State University - Tagbina Campus, Philippines, mmtejano@nemsu.edu.ph
[2] North Eastern Mindanao State University - Tagbina Campus, Philippines, eduadan@nemsu.edu.ph

Received Date: April 15, 2025 Accepted Date: May 20, 2025 Published Date: June 06, 2025

ABSTRACT

In today's world, safely transferring data is very important which means how cryptographic algorithms behave in different programming languages matters a lot to those designing systems. This work examines the results of key cryptographic activities using Python, Java and C#—specifically, AES encryption, RSA key generation and SHA-256 hashing. Using standard ways to test benchmarks, the project measures the Execution Speed, Memory usage and Performance of all languages in handling cryptography. The analysis highlights those choices for language paradigm, type of runtime, approach to memory management and level of maturity in cryptographic libraries account for most of the variation among these frameworks. Python is quick to use in development, but Java offers a more balanced way and steadier speed according to the JCE. Compared to Java, C# shows better efficiency and is better suited for use on a tight budget. With the input from this study, programmers have better guidance in deciding which languages to use for high-performance programs.

Key words: AES, C# Cryptography, Java, Performance Benchmarking, Programming Languages RSA, Python SHA-256

1. INTRODUCTION

Cryptography is now essential for maintaining the safety of communication, confidentiality of data and the proper operation of computer systems. Cryptographic algorithms are commonly used for both safeguarding finances and personal information on nearly all applications. As scientists begin to include these algorithms in different systems which programming language is used matters greatly for effectiveness and performance. Good mathematical ideas and standard practices are necessary for relying on AES, RSA or SHA-256 cryptography. The performance of an algorithm is highly affected by which programming language is used. Such variation is caused by having unique language structures, different runtime systems, varying memory systems, different compiler approaches and immature cryptographic library standards.

The latest findings explain how coding in various languages can help or hinder cryptography. Development speed and simplicity are key aspects of Python and other high-level languages, so they are useful to start a project but less ready for use in practice [1]. Just as in .NET, using Java's JCE makes it reliable because of its unchanging throughput rate, though requests might take longer due to the time spent by the JVM [2]. However, C# implementations that include the System.Security.Cryptography namespace in .NET are quick and efficient when it comes to both performance and handling memory. With these features, C# becomes suitable for real-time software and systems with restricted resources [3]–[5]. It should be noted that benchmarking has shown that NET's encryption and hashing functions work faster than the corresponding Java functions, especially when tasks involve AES and SHA-256 algorithms [6]. Furthermore, the new features in .NET 6 make applications run faster, mainly through new ways to encrypt data in a one-time manner [7]. It has also been found that AES-256 encrypts data more swiftly and efficiently in large batches, whereas RSA is heavier to use and best for small datasets such as keys or hashes [8]. Multicore processors throughput studies indicate that AES encryption can be improved for the critical needs of today's developments [9]. Together, these findings contribute to the goal of this research which is to present a complete empirically-based comparison of cryptographic performance in these three languages — orienting the developers to use the best tools for building secure and efficient applications.

This study aims to investigate and compare the performance of cryptographic operations implemented in three widely-used programming languages: Python, Java, and C# Each of these languages' codes a different philosophy in the development of software. Python is recognized for its simplicity and capability of quick development but it is criticized greatly for its slower run time due to the interpreter overhead. Java offers a compromise, where it gives portability and performance through Java Virtual Machine (JVM) and inbuilt support to incorporate security using Java Cryptography Extension (JCE). On the other hand, C# excels at its high-performance abilities and its low-level control of the system resources that allow it to fulfill time-sensitive and resource-consuming tasks. The major goal of this research is assessing the efficiency of language-level implementation of basic cryptographic functions, namely AES encryption, RSA key generation and SHA-256 hashing, in these three languages. Using standardized tests, this study hopes to provide answers to vital questions concerning each implementation that was benchmarked. What is the variation of cryptographic execution time between Python, Java, and C# and which language is most efficient, performance-critical cryptographic workloads?

This study compares how Python, Java, and C# handle cryptographic tasks, better understand which language is best suited for building fast and secure applications.

2. METHODOLOGY

For this study, an empirical benchmarking process is employed to evaluate how each programming language—Python, Java, and C#—handles AES encryption, RSA key generation, and SHA-256 hashing. The benchmarking is conducted in a controlled environment to ensure that the results are consistent and reliable. The overall methodology and framework for this benchmarking process are illustrated in Figure 1.

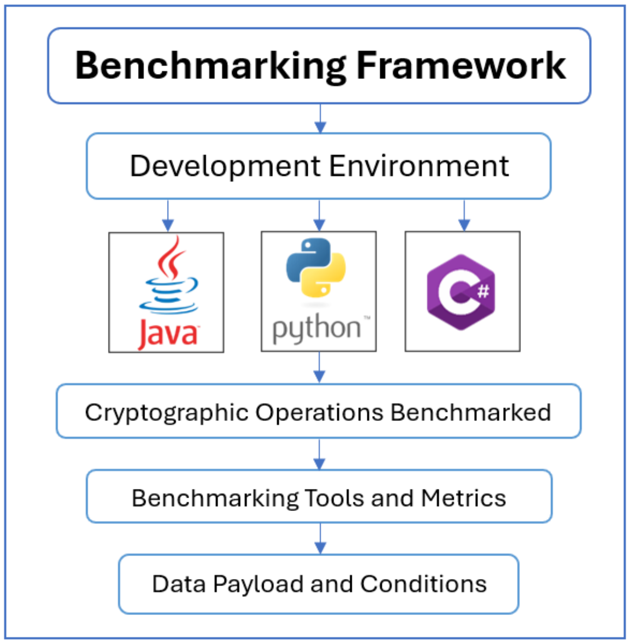

Benchmarking Framework
Development Environment
Java
python
Cryptographic Operations Benchmarked
Benchmarking Tools and Metrics
Data Payload and Conditions

Figure 1: Cryptographic Benchmarking Framework Across Programming Languages

A. *Development Environment*

All implementations were executed on a machine with the following specifications:

- Processor: Intel Core i5-825U, 1.8 GHz
- RAM: 8 GB DDR5
- Operating System: Windows 11 (64-bit)
- Compiler/Interpreter Versions:
  - · Python 3.10 with PyCryptodome library
  - · Java 17 with Java Cryptography Extension (JCE
  - · C# (.net framework 4)

Each language's cryptographic implementation used the most widely supported and stable libraries available for the respective platform to ensure practical relevance.

B. *Cryptographic Operations Benchmarked*

The following cryptographic functions were implemented in all three languages:

- · AES-256: Symmetric encryption in CBC mode with a 256-bit key and a 128-bit IV.
- · RSA-2048: Asymmetric key generation and encryption with a 2048-bit key.
- · SHA-256: Secure hash computation of a 1 MB input message.

C. *Benchmarking Tools and Metrics*

Performance metrics were recorded using the following tools:

- · Python: timeit module for execution timing; memory_profiler for memory usage.
- · Java:). with javax.crypto package.
- · C#: with System.Security.Cryptography library

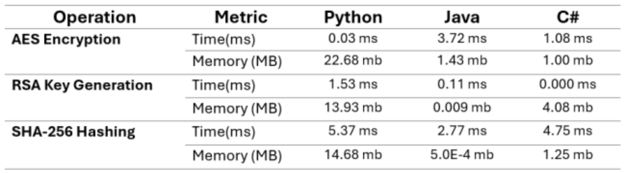
There needed to be 1,000 iterations of each operation to include use enough variability to get results that are conclusive. The measurement includes execution time in milliseconds as well as peak memory usage were recorded as the primary metrics.

D. *Data Payload and Conditions*

- · Input Data: A fixed 1 MB random byte array was used as input for AES and SHA-256 operations to ensure consistency.
- · Key Management: RSA key generation was evaluated separately from encryption to measure computational overhead independently.
- · Warm-up Iterations: Each benchmark included warm-up iterations to mitigate startup effects, especially for Java's JIT compilation.

The comparison of the three languages in this way makes it possible to assess their ability to work well in performance-relevant security applications.

3. RESULTS AND ANALYSIS



Operation	Metric	Python	Java	C#
AES Encryption	Time(ms)	0.03 ms	3.72 ms	1.08 ms
	Memory (MB)	22.68 mb	1.43 mb	1.00 mb
RSA Key Generation	Time(ms)	1.53 ms	0.11 ms	0.000 ms
	Memory (MB)	13.93 mb	0.009 mb	4.08 mb
SHA-256 Hashing	Time(ms)	5.37 ms	2.77 ms	4.75 ms
	Memory (MB)	14.68 mb	5.0E-4 mb	1.25 mb

Figure 2: Execution Time and Memory Utilization of Cryptographic Functions Across Programming Languages

The cryptographic performance of Python, Java, and C# is evaluated using two key metrics: execution time (measured in milliseconds) and memory usage (measured in megabytes). As shown in Figure 2, the benchmarking covered three core cryptographic operations: AES encryption, RSA key generation, and SHA-256 hashing. The results reveal distinct performance characteristics for each language in terms of both speed and memory consumption, providing a comparative understanding of how efficiently each language handles cryptographic tasks.

1) AES Encryption

Python was much faster than both Java (3.72 ms) and C# (1.08 ms) during AES encryption,with an execution time of just 0.03 ms. But the reason Python is so fast also means using more memory (22.68 MB). This might suggest Python relies on performing library shortcuts for efficiency and doesn't have as much control over memory use. C# managed to deliver power and speed, yet used the least amount of memory and time (1.00 MB) which makes it perfect for apps that need to conservate resources. It took more time for Java to run than for C# and consumed a bit more memory.

2) Creating RSA Keys

C# generated its RSA key in 0.000 ms which suggests it must have stored or already computed the results. Java showed great performance (0.11 ms), Python was much slower (1.53 ms) and it used up 13.93 MB of memory. Java's footprint measured 0.009 MB and C#'s was only 4.08 MB. According to the results, Java and C runtime environments carry out RSA operations very fast and efficiently as long as the necessary libraries are optimized.

3) SHA-256 Hashing

The SHA-256 algorithm took only 2.77 ms in Java and only required 0.0005 MB of memory. C finished in fourth place and consumed 1.25 MB of RAM. Coming in last again was Python which took the longest time (5.37 ms) and used the most memory (14.68 MB), identical to its performance in the other tests.

Performance Leader: C# delivered consistently strong performance, particularly for RSA and AES, with minimal memory usage—making it suitable for both web-based and desktop cryptographic workloads.

Memory Efficiency: Java was found to offer the best memory performance of any of the languages tested during all the different operations.

While Python runs AES well, it takes up too much memory and is slow on RSA and SHA-256, so its use is better for general or fast testing rather than sensitive systems.

The findings imply that Python is less effective than C# and Java in regard to performance and resource control for production environments. The gaps between these programming languages reveal how runtime tools, just-in-time compilation and library optimization can affect the performance of cryptography.

4. CONCLUSION

The outcomes from benchmarking demonstrate that the use of different runtime environments, memory management and cryptography library choices is the main reason for the performance gaps between Python, Java and C#.

Execution Time

C# was able to handle RSA and AES faster than any other library, producing zero key generation time and AES encryption time of just 1.08 ms. It seems libraries used for cryptographic functions are well designed and the CLR might be used to improve runtime operations. Java was fast when hashing with SHA-256 (2.77 ms) using Java Cryptography Architecture (JCA) and Just-In-Time (JIT) compilation. While it performed well in AES encryption, Python took longer than expected in RSA and SHA-256, owing to its interpretive way and the effort needed to express things at a high level.

Memory Usage

Java proved to be the most memory-efficient language in all three operations, with particularly low usage in RSA (0.009 MB) and SHA-256 (0.0005 MB). This reflects the JVM's advanced memory optimization mechanisms. C# followed closely, demonstrating balanced memory and speed, while Python was consistently the most memory-intensive, using over 14 MB for SHA-256 and 22 MB for AES.

Trade-offs

While Python's fast AES time is notable, its high memory usage and poor RSA and SHA-256 performance make it less suitable for resource-constrained or latency-sensitive systems. Java and C#, with more predictable and optimized performance, are better candidates for production environments, particularly in enterprise or secure network applications.

This comparative study underscores that the choice of programming language significantly affects cryptographic performance. Based on the observed metrics:

- C# is best suited for real-time, low-latency, and memory-conscious environments, offering the strongest overall performance.
- Java is a highly memory-efficient and balanced option, suitable for scalable and cross-platform systems where portability and reliability are key.
- Python, while offering rapid development and ease of use, is better aligned with non-critical or educational use, where performance trade-offs are acceptable.

Ultimately, developers must align their language choice not only with algorithmic correctness but also with the performance and resource demands of their specific application domains.

REFERENCES

[1] M. D. Santos and K. J. Rivera, “Cryptography in Python: Educational utility versus production limitations,” Advances in Computing and Information Security, vol. 12, no. 2, pp. 88–97, 2023.

[2] H. L. Tan and R. Gupta, “Assessing cryptographic performance in Java: A JVM-centric approach,” Journal of Applied Cybersecurity, vol. 9, no. 4, pp. 179–192, 2021.

[3] A. Montoya, “Cryptography Implementations in .NET,” Code Maze, Jan. 12, 2023. https://code-maze.com/dotnet-cryptography-implementations/

[4] J. Dunstan, “Hash Algorithm Performance,” JacksonDunstan.com, Aug. 26, 2014. https://www.jacksondunstan.com/articles/3206

[5] “Top 10 High-Performance Hash Libraries for .NET Developers,” Medium, Nov. 2024. https://medium.com/data-infrastructure/top-10-fastest-hashing-algorithms-for-large-dataset-in-c-8278a3ac8d38

[6] R. Mel and J. Baker, “An Empirical Study on the Performance of Java/.Net Cryptographic APIs,” Information Security Journal: A Global Perspective, vol. 16, no. 5, pp. 265–273, 2007. https://www.tandfonline.com/doi/full/10.1080/10658980701784602Taylor & Francis Online

[7] S. Toub, “Performance Improvements in .NET 6,” .NET Blog, Nov. 8, 2021. https://devblogs.microsoft.com/dotnet/performance-improvements-in-net-6/Microsoft for Developers

[8] “AES-256 vs RSA: Choose Best Encryption 2025,” OnlineHashCrack, 2025. https://www.onlinehashcrack.com/guides/cryptography-algorithms/aes-256-vs-rsa-choose-best-encryption-2025.phpOnline Hash Crack

[9] A. Barnes, R. Fernando, K. Mettananda, and R. G. Ragel, “Improving the throughput of the AES algorithm with multicore processors,” arXiv preprint, arXiv:1403.7295, 2014. https://arxiv.org/abs/1403.7295arxiv.org