SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V13I9P105

Volume 13 Issue 9, 55-65, September 2026

Original Article

Reproducible AES-128 Benchmarking and Security-Assessment Boundary for Resource-Constrained IoT Environments

Manisha Ahirrao[1], Bageshree Pathak[2], Mrudul Dixit[3]

[1,2,3]*Department of Electronics and Telecommunication, MKSSS's Cummins College of Engineering Pune, Maharashtra, India.*

[1]*Corresponding Author : mnshahirrao25@gmail.com*

Received: 01 June 2026 Revised: 28 August 2026 Accepted: 02 September 2026 Published: 26 September 2026

Abstract *- To meet the constraints of resource-constrained Internet of Things (IoT) devices, the encryption mechanism must provide confidentiality while adding little complexity and memory usage. This work presents a reproducible Advanced Encryption Standard (AES) -128 performance benchmarking and security boundary analysis for constrained IoT applications. Before any benchmarking, the AES-128 implementation was tested against a standard cryptographic test vector. Encryption and decryption were verified based on the successful plaintext recovery. To enhance methodological rigour and reproducibility, the experimental design used 250 runs over four IoT-relevant payload sizes (16 bytes, 1 KB, 10 KB, and 1 MB) executed in a controlled execution environment. The evaluation consisted of performance metrics that include encryption time, decryption time, execution stability, standard deviation and memory variation, analysed in a controlled execution environment. Results show that while AES-128 exhibits stable run-to-run timing for small and medium payloads, execution overhead and timing variance increased for 1 MB workloads in the implementation presented using Python. It also points out that process-level memory profiling is limited for small workloads since they are impacted by noise and variability in the measurement. Reduced-key brute-force attack analysis is shown only as a pedagogical example of key-space growth and makes no claim about the security of full AES configurations. Likewise, side-channel attack concerns are qualitatively addressed because attack analysis was performed. The study provides an AES-128 benchmarking methodology for IoT security evaluation that is statistically founded and fully reproducible.*

Keywords *- AES-128, IoT security, Performance evaluation, Reproducible benchmarking, Reduced key demonstration.*

1. Introduction

Cryptographic algorithms were once confined to servers, desktop systems and high-end mobile platforms. They now execute routinely on sensor nodes, wearable devices, smart meters, embedded controllers and industrial telemetry systems. Such devices handle sensitive data while operating under severe limits on computation, memory, energy and software complexity. The research question is therefore no longer whether encryption is required, but how to deliver cryptographic protection at minimum overhead without overstating the security obtained or understating its cost. The AES remains among the most widely deployed symmetric cipher for securing communication and stored data [1]. It uses a fixed 128-bit block with variable key sizes (128, 192 and 256 bits), respectively, where longer keys require additional transformation rounds. In real systems, AES operates within a block cipher mode of operation such as ECB, CBC, CFB, OFB or CTR [2], which defines how messages longer than a single block are processed [3]. Suitability for constrained IoT nodes therefore depends not only on the algorithm but on the implementation strategy, mode selection, key management, leakage resistance and device-specific resource limits. Conventional implementations can impose prohibitive area, latency, memory or power costs on ultra-low-power devices.

To address this, the US National Institute of Standards and Technology initiated a lightweight cryptography standardisation programme for platforms where conventional primitives are not efficiently deployable [3, 4], concluding with the ASCON family for compact authenticated encryption and hashing. Designs such as PRESENT, SIMON and SPECK [5] and software-oriented ciphers such as ChaCha20 [6] target the same constraints. Depending on the application, neither AES nor lightweight cyphers is always better. Cipher selection remains a design decision governed jointly by the threat model, the hardware platform and the software architecture. Figure 1 shows the hybrid deployment arrangement assumed in this work, in which a constrained sensor node, an intermediate gateway and a cloud endpoint each apply a different cipher configuration.

Three methodological gaps persist across this literature.

- Reported performance figures frequently derive from single-run measurements, so no dispersion is available and run-to-run stability cannot be assessed.
- Cross-algorithm comparisons are assembled from measurements taken on different processors, compilers and language runtimes, which makes the resulting rankings difficult to reproduce and unsafe to generalise.
- Performance claims and security claims are often merged into a single argument, so that a reduced-key demonstration intended for teaching can be read as evidence about the strength of a full AES configuration.

The problem addressed here is therefore not whether AES-128 is fast enough for IoT deployment, but how an evaluation of AES-128 can be reported so that an independent group can reproduce it and so that the boundary of what the measurements support is stated explicitly.

The contributions are fourfold. Firstly, a controlled benchmarking framework built on a validated AES-128 testbed. Secondly, a repeated-execution protocol comprising 250 runs across four IoT-relevant payload sizes. Thirdly, a statistical assessment covering execution time, scalability behaviour, memory consumption and implementation-specific limits. Fourthly, an explicit security-claim boundary separating measured AES-128 performance from theoretical key-space assumptions, from the reduced-key demonstration, and from qualitative side-channel threat modelling.

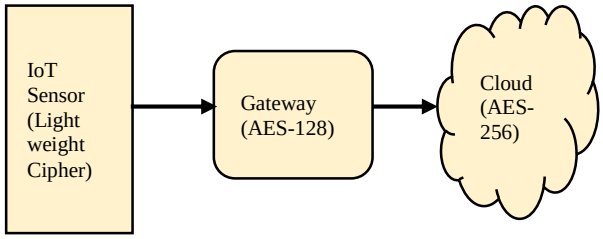

Fig. 1 IoT hybrid encryption deployment architecture

2. Literature Review

Recent comparative evaluations agree that AES-128 remains a viable baseline on constrained hardware, though not the most efficient option. Radhakrishnan et al. [7] compared AES-128, SPECK and ASCON on Arduino boards across execution time, memory utilisation, latency and throughput, and reported SPECK as the strongest performer under constraint. El-hajj et al. [8] reached comparable conclusions at considerably larger scale, benchmarking 39 block ciphers on an ATMEGA328P microcontroller before extending the comparison to NIST second-round candidates. Both studies treat AES-128 as the reference point rather than as the recommendation. The ranking is not stable across measurement contexts. Silva et al. [9] evaluated cryptographic configurations inside HTTP and MQTT stacks on six embedded devices and found that transport and protocol choice materially affected message delay and overhead. Sorescu et al. [10] observed different orderings for the same set of ciphers under a low-data-rate Bluetooth mesh scenario and under bulk transfer on identical hardware. A performance ranking is therefore a property of a measurement configuration rather than of an algorithm. Standardisation of ASCON has produced a body of platform-specific evaluation, represented by Sarasa Laborda et al. [11], which covers the ASCON family across four Arduino platforms and observes that prior work reports theoretical metrics or measurements from hardware-accelerated platforms rather than results from commodity devices. Standardisation has settled which primitive to reach for without settling how a claim about its performance should be reported.

Three reporting practices recur across these studies. Repetition counts are small or unstated. Execution environments are described at varying levels of completeness. Performance evidence and security assumptions are presented within a single argument. The present study addresses these three practices directly rather than proposing a new cipher.

2.1. AES as a Standardised Symmetric Block Cipher

AES is an established symmetric-key encryption algorithm in the form of a substitution permutation network. AES-128, AES-192 and AES-256 use ten, twelve and fourteen transformation rounds, respectively. This would be dependent on the key length selected. This directly influences how large the brute-force search is, and the shape of a round with respect to its capacity to provide diffusion, confusion and general robustness against normal attacks. But on real-life deployments, AES is very rarely used as a single-block primitive alone. In fact, it is generally employed within a block cipher mode of operation, or an authenticated encryption. This is an important distinction, as the vast majority of real-world security failures are a result not of flaws in the AES algorithm [13] but poor mode choice or poorly chosen authentication mechanisms or information leakage at the implementation level. Which block cipher mode is selected also has an impact on both security implications and performance patterns. ECB mode produces the same ciphertext byte or bytes for any identical plaintext block, leaking structural information and implying it cannot be used in most real-world applications. Chaining the ciphertext blocks in CBC mode and using random initialisation vectors helps alleviate this weakness with semantic security. Other modes are similar in that they support stream-oriented operation (i.e., CFB, OFB and CTR) but have different properties for synchronisation and error-propagation [2]. A methodological difference relates to measurements of AES key length. AES and its variants are defined standard configurations with fixed and formally specified key sizes.

2.2. Lightweight Cryptography and IoT Constraints

Researchers consider lightweight cryptography as cryptographic algorithms and constructions designed for applications with constrained hardware resources in terms of area, memory, energy or latency. Standardisation of lightweight cryptography has been initiated by NIST in response to implementation environments where traditional cryptographic primitives may not be appropriate for efficient deployment due to limited functionality at resource-constrained devices [12]. The later standardisation of an ASCON-based lightweight cryptography is a major step forward, by offering a standardised authenticated encryption for resource-constrained environments. Yet, standardisation by itself does not remove the need for benchmark and implementation analysis. It is entirely possible for an algorithm that behaves well on one hardware platform to perform much worse within another software or architectural environment. PRESENT [13] is a lightweight block cipher with an ultra-small area due to its very small number of logic gates, so that it can be adapted for RFID systems or compact sensor networks. Later lightweight cipher families, SIMON and SPECK, were also proposed to allow multiple implementation flexibility in different word sizes, block sizes, and computational environments. On the other hand, ChaCha20 is different in structure compared to lightweight block ciphers since it is a stream cipher. It is widely used as an efficient software-oriented encryption scheme and serves as the primitive building blocks for many standardised authenticated encryption constructions. These examples demonstrate that lightweight cryptography is a heterogeneous collection as opposed to a comparability class of algorithms. Depending on the implementation environment and measurement methodology, ranking with respect to evaluation criteria such as hardware area, RAM consumption, code size, throughput, energy efficiency, and resistance to side-channel leakage can differ substantially [14]. Thus, scientifically valid relative comparisons of different algorithms cannot combine measurements derived from implementation assumptions that vary between those algorithms and present them as definitive or reproducible cross-algorithm performance rankings. AES configurations show that different key sizes and mode selections have effects on performance characteristics, but the magnitude of these effects is highly dependent on hardware architecture, compiler behaviour and software design details. The different applied studies on banking security, ethical hacking, cyberattack countermeasures and information assurance indicate that encryption is merely a sub-component of the much larger information assurance ecosystem. It includes various components like authentication/identity verification, access control and safeguards such as continuous monitoring and operational controls [15]. These studies can give context, but providing claims for quantitative performance should not replace controlled experimental benchmarking. Related work on password cracking with and without a privacy-preserving approach [16] and on the adoption of cryptographic software further confirms that security does not necessarily depend only on cryptographic algorithms but also, at least in terms of protocols, users, implementation-level protections, and operational practice.

2.3. Side-Channel Attacks and the Need for an Explicit Threat Model

Early research by Kocher about timing attacks [17] showed that the secret key could be obtained by observing execution time. Later research also demonstrated that cache-timing and cache-collision attacks can be leveraged against AES implementations based on table lookup or dependent memory-access patterns. In this study, the nature of the attack discusses side-channel exposure solely as a deployment-level security concern for constrained IoT devices, where potential vulnerabilities can stem from table-based AES implementations, branch-dependent execution paths, shared-cache architectures or lack of sufficient masking techniques. The manuscript also recommends the use of constant-time implementations, authenticated encryption methods, implementation-level hardening techniques and device-specific validation before use in security-critical or high-risk operational contexts.

2.4. Research Gap Analysis

Table 1 positions the present study against the representative work discussed in Sections 2.1 to 2.3. The comparison is deliberately restricted to methodological attributes, because absolute performance values obtained on different platforms are not directly comparable and tabulating them side by side would reproduce the very practice criticised above. Three attributes separate this study from the reviewed literature: the number of repeated executions performed per condition, the reporting of dispersion alongside central tendency, and the presence of an explicit statement separating measured performance from assumed security properties. None of the reviewed studies reports all three together.

Table 1. Methodological positioning of the present study against the reviewed literature

Methodological aspect	Typical treatment in reviewed literature	Treatment in the present study
Repetitions per condition	Single run, or a small number left unstated	250 executions for every payload size
Dispersion reporting	Mean or best-case value only	Mean, standard deviation, histogram and time series
Payload coverage	One representative payload size	16 B, 1 KB, 10 KB and 1 MB

Execution environment	Frequently unstated or partially stated	Single fixed host, fixed key and fixed plaintext
Correctness validation	Rarely reported before benchmarking	Test vector check and round-trip check done before benchmarking
Security claim scope	Performance and security claims merged	Explicit boundary stated

3. Experimental Design

AES-128 was selected because it remains one of the most widely deployed standardised symmetric ciphers and provides a stable baseline against which lightweight cryptographic schemes can be compared in future work. First, a standard cryptographic test vector is used to validate the correctness of the algorithm by simulating an AES-128 implementation. The benchmarking only starts if the generated ciphertext is equal to the reference output expected. Then, a round-trip validation consisting of encrypting the plaintext and decrypting its ciphertext is done to ensure the recovered plaintext matches the original input. Then correctness-checking and benchmarking are done by invoking this on four input sizes of 16 bytes, 1 KB, 10 KB and 1 MB.

These payload sizes represent a spectrum of communication scenarios relevant to IoT, from telemetry messages produced in single-block mode through buffered data. For the same key length, the same input data, and a fixed payload size, the same encryption and decryption operations were performed with 250 run count. The time taken for encryption, decryption, average execution time, and standard deviation were the performance metrics considered. Mean execution time is the average runtime observed for the implementation under investigation, and the standard deviation quantifies the run-to-run variation, indicating the stability of execution. An Intel Core i5 processor with 8 GB RAM was used, and the implementation environment was Python, as defined within the experimental setting. For each of the payload sizes examined, the benchmarking workload remains consistent. To make sure that at no point random data generation is inadvertently adding to timing measurements, a fixed cryptographic key and fixed plaintext input are always used across repeated benchmark execution. This is an experiment not to test randomness quality, but instead to measure the runtime stability under fixed-input-length circumstances.

3.1. Statistical Metrics

To verify the computational scalability in a resource-constrained environment, all experiments were conducted under controlled software conditions. To reduce measurement variability, execution time was recorded using a high-resolution timer and averaged over multiple independent runs. Throughput is calculated and indicates the number of encryption and decryption operations executed within a given time [15]. Encryption and decryption throughput are computed as:

$$Throughput = \frac{Plaintext\ size(bytes)}{Execution\ Time(Sec)} \quad (1)$$

3.2. Brute - Force Attack - Scope and Security Interpretation

A controlled brute-force attack model is developed to evaluate the performance of the proposed AES implementation. This designed model will exclusively search the reduced key space described below [8]. Due to the computational infeasibility of exploring the complete 128-bit AES key space 2^{128} under practical experimental constraints, a reduced key-space configuration was employed. This approach enables systematic evaluation and preserves the theoretical consistency with standard cryptographic principles. Let K denote the 128-bit AES secret key. Assume that a portion of the key K_{known} is known while the remaining portion $K_{missing}$ consists of n unknown bytes. The size of the reduced key space is therefore: 2^{8n} Since each byte can assume 256 possible values.

The number of unknown bytes is defined as:

$$n = length\ (K) - length\ (K_{known}) \quad (2)$$

Let the ciphertext be given as:

$$C = (c\_1, c\_2, \dots, c\_m) \quad (3)$$

A counter variable $Count = 0$ is initialised to track the number of key attempts. The brute force procedure iterates over all possible candidates in the reduced key space:

$$K \in \{0,1, \dots, 2^{(8n)} - 1\} \quad (4)$$

Construct $K_{missing}$ as the n-byte representation of K. Form the complete key:

$$K = K_{known} \parallel K_{missing} \quad (5)$$

Where "‖" denotes concatenation. Perform decryption:

$$P' = D_K(C) \quad (6)$$

Compare the decrypted output P' with the known plaintext (or a predefined verification condition). If the generated key matches the correct one, the search stops because the right key has been found. If not, the system simply increases the counter and keeps checking until a match is discovered. For a n bit length key, the total number of possible keys is:

$$|K| = 2^n \quad (7)$$

In the worst case, an attacker must evaluate all 2^n possible keys. Assuming uniform key distribution, the expected number of trials required to recover the correct key is:

$$T_{attack} = 2^{(n-1)} \quad (8)$$

The key space size is 2^{128} for standard AES-128, which renders exhaustive search computationally infeasible in the current technology. For experimental feasibility, a reduced key space was employed in this study. With respect to key length, the brute force complexity was measured under a controlled configuration.

By using CrypTool, a partial key exposure model was implemented to evaluate the resistance of AES-128 against key search. Rather than attempting a full 2^{128} key search (which is computationally infeasible), a controlled brute-force experiment was conducted by assuming that a portion of the key is known and restricting the exhaustive search to the remaining unknown bits.

Let the total AES key length be 128 bits. If k bits are assumed to be known, the number of unknown bits is:

$$u = 128 - k \quad (9)$$

The effective brute force search space is therefore:

$$\left|K_{effective}\right| = 2^u \quad (10)$$

For each configuration, AES encryption was performed using CBC, ECB and CFB modes. A known plaintext attack model was assumed in which candidate keys were iteratively tested until the decrypted ciphertext matched the known plaintext. Execution time was measured from the start of the brute force process until successful key recovery.

Each experiment was repeated multiple times, and the reported values represent average recovery times. The objective of this setup is not to compromise AES-128 in practice, but to empirically validate the exponential scaling property of brute-force attacks under increasing key entropy. It is important to note that differences observed across ECB, CBC and CFB modes are attributed to implementation-level operational overhead and internal processing characteristics of CrypTool. These variations do not imply differences in brute force security strength, as the theoretical complexity remains solely dependent on the effective key space size.

3.3. Memory Profiling and Timing Model

The memory usage of AES was measured for different key sizes to understand how its internal structure affects storage requirements. Memory measurements were obtained using Python runtime profiling tools under identical execution conditions. Values represent average observed allocation during encryption of 1 MB plaintext.

The computational time of a cryptographic algorithm can be modelled as a function of key size and number of rounds. For a given hardware platform, execution time may be expressed as:

$$T(k, r) = c_1 \cdot r \cdot f_{round}(k) \quad (11)$$

Where c_1 represents a constant dependent on hardware efficiency r denotes the number of encryption/decryptions round and $f_{round}(k)$ characterises the computational complexity per round for a given key size k. Since the number of rounds is directly proportional to the total number of substitution, permutation and key mixing operations, computational time increases approximately linearly with r under fixed architectural conditions. When the number of rounds is reduced for performance optimisation, the modified execution time can be expressed as:

$$T_{modified}(k, r') = c_1 \cdot r' \cdot h_{round}(k) \quad (12)$$

Where $r' < r$ and $h_{round}(k)$ represents a simplified per-round computational structure. Because r' is smaller, the overall execution time decreases proportionally, leading to improved throughput and reduced latency.

This approach is often considered in constrained platforms such as IoT devices and embedded controllers where computational resources and energy availability are limited. Round reduction decreases computational overhead and energy consumption, which can be attractive for highly constrained IoT devices.

4. Results and Discussion

Table 2 reports the mean encryption and decryption times for each payload size.

Table 2. AES-128 encryption and decryption timing versus input payload size

Input size	Encryption mean (s)	Encryption SD(s)	Decryption mean (s)	Decryption SD(s)
16 B	Less than 0.01	Less than 0.01	Less than 0.01	Less than 0.01
1 KB	Less than 0.01	Less than 0.01	Less than 0.01	Less than 0.01
10 KB	0.358	0.008	0.68	0.006
1 MB	53.2	24.5	94.0	92.0

Figure 2, AES-128 encryption time for a 1 MB input payload, shows both a significant increase in execution time and a significantly greater range of error magnitudes than the conditions with smaller payloads. These observations highlight the need for implementation optimisations, use of established cryptographic libraries, effective streaming architectures, hardware accelerators and benchmarking in the target deployment platform as viewed from an embedded-systems viewpoint. Figure 3 shows AES-128 decryption time for the same range of payload sizes. The 1 MB workload shows not only a higher mean runtime but also more timing dispersion than with the medium-sized and small payload conditions.

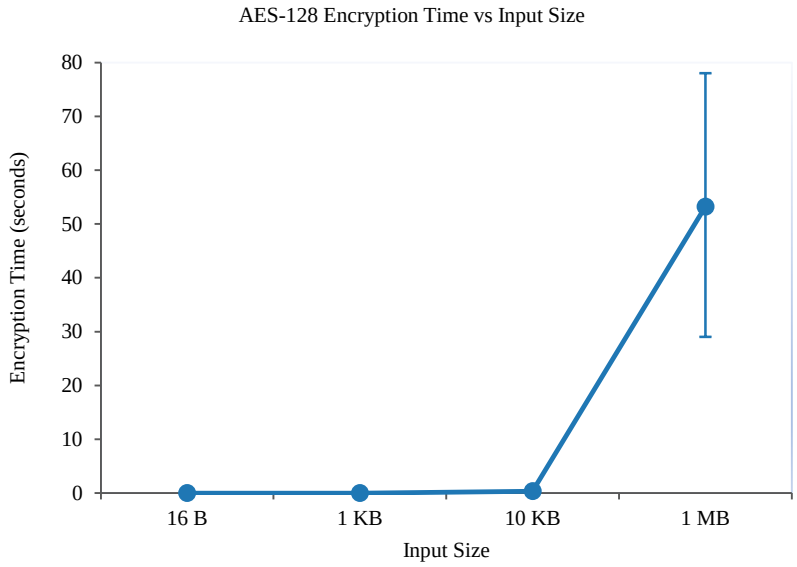

Fig. 2 AES-128 encryption time vs. input size

In the measurements reported here, decryption is slower than encryption. This is expected for table-based software implementations, because the inverse Mix-Columns transformation costs more than the forward transform. Hardware support such as AES-NI reduces this difference.

The memory delta for each payload size tested can be seen in Figure 4. The largest positive memory change occurs in the 1 MB workload, due to buffering overhead and intermediate object allocation.

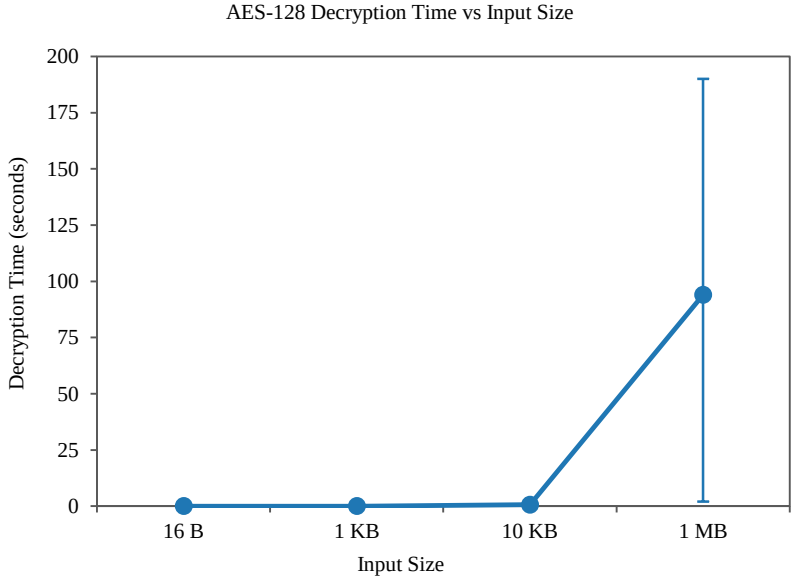

Fig. 3 AES-128 decryption time versus input size

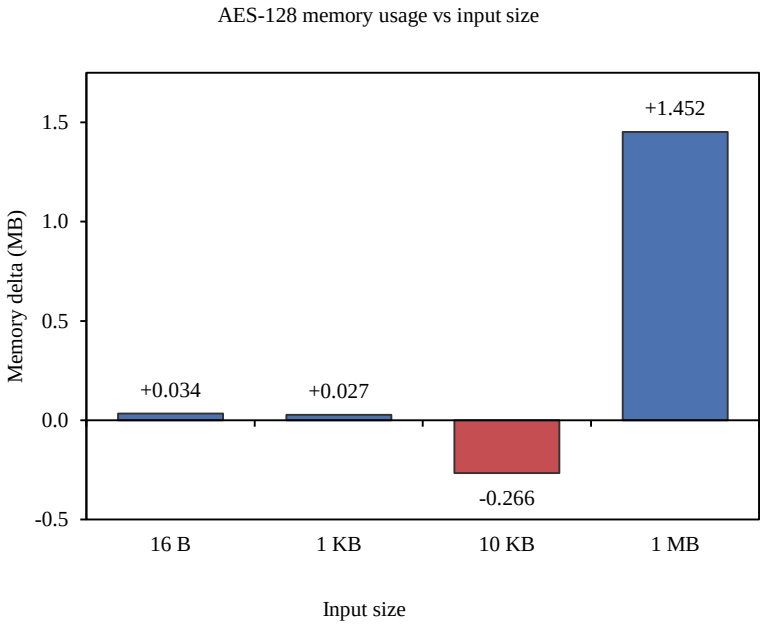

Fig. 4 AES-128 memory delta versus input size

Meanwhile, the results show that smaller payload conditions have almost no memory delta, whereas the 10 KB workload has a negative memory delta. This negative process-level memory delta should not be read as evidence that the encryption algorithm consumes negative memory resources. Instead, it captures the effect of memory allocation and deallocation taking place in the interpreter and OS environment over the measurement window. The 10 KB workload makes a good compromise for runtime stability analysis because it is small enough that large-scale buffering and memory-management effects do not completely overshadow any potential optimisations while still being large enough to go beyond the triviality of an operation on a single block.

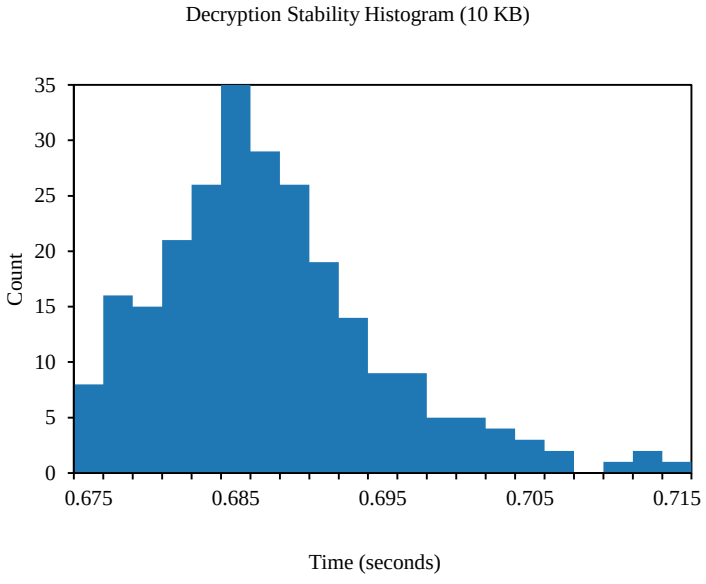

Fig. 5 Decryption stability histogram for the 10 KB workload over repeated executions

The decryption-time histogram for the 10 KB workload is depicted in Figure 5. Out of the 250 observations, 246 fall within the central distribution range, and the remaining four are slower executions.

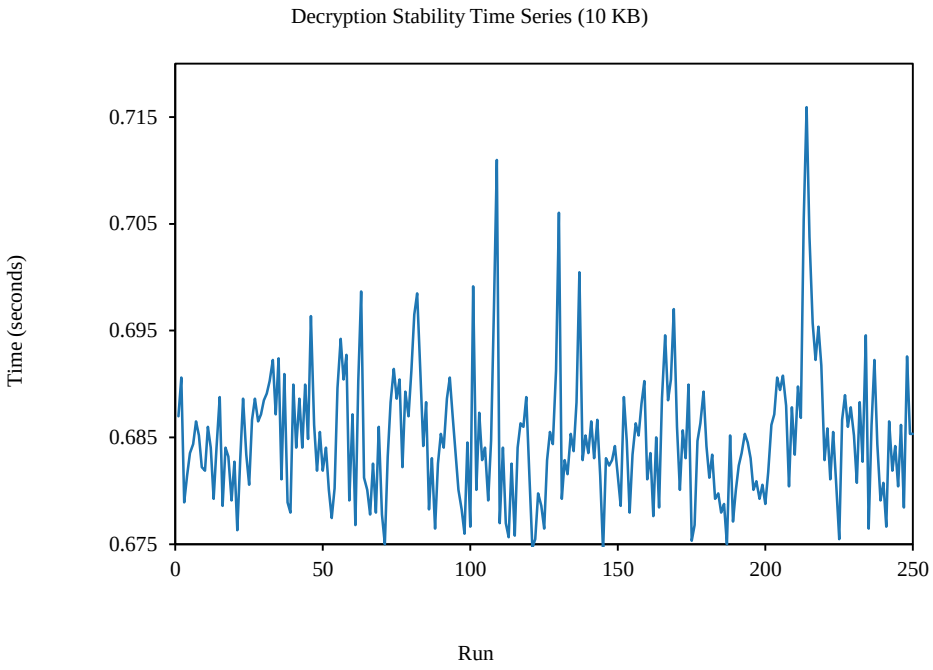

Fig. 6 Decryption stability time series for the 10 KB workload

The results of the time-series analysis for decryption, as illustrated by Figure 6, demonstrate that some timing spikes were still observable, but a majority of executions remain stable between runs. Figures 7 and 8 show the encryption stability for the same workload. As seen from the histogram of encryption times, the encrypted time measurement exhibits a close clustering feature.

When plotted on a corresponding time series visualisation, the execution behaviour is seen to stabilise with some slight exceptions, noted by one large-time spike. Most encryption measurements are tightly clustered, with 88 per cent of runs falling within two adjacent bins, although the rare excursions are larger in relative terms than those observed for decryption.

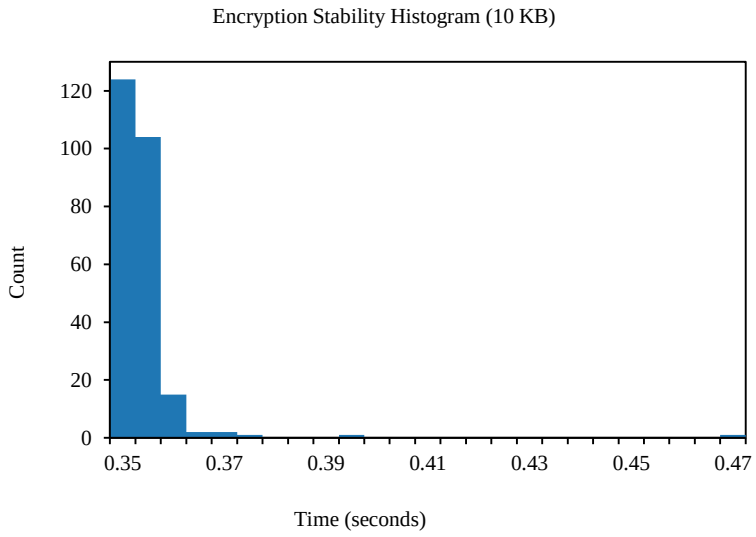

Fig. 7 Encryption stability histogram for the 10 KB workload over repeated executions

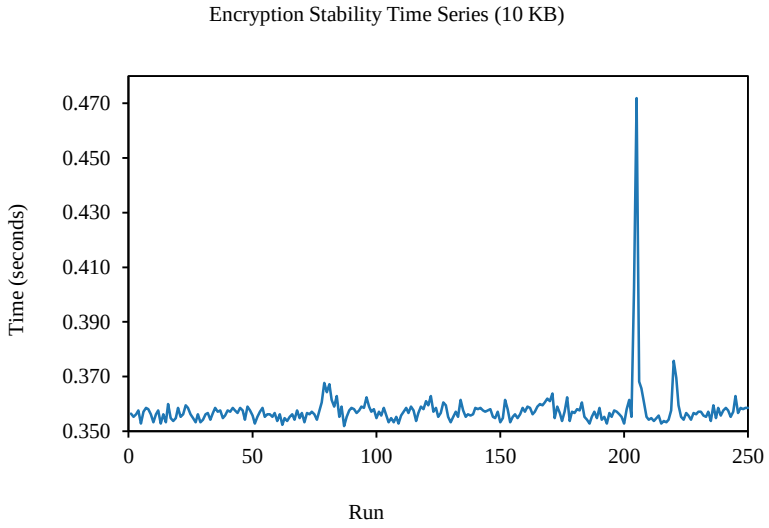

Fig. 8 Encryption stability time series for the 10 KB workload

Table 3. Evidence-supported interpretation of the AES-128 results

Result component	Evidence observed	Permissible interpretation
Encryption scalability	Runtime is very small for 16 B, 1 KB, and 10 KB on the plotted scale but rises sharply at 1 MB.	The evaluated Python AES-128 implementation scales poorly for large payloads compared with small telemetry-style payloads.
Decryption scalability	The 1 MB decryption case shows high runtime and wide dispersion.	Large-payload decryption results are affected by implementation and system noise. They should not be generalised to optimised AES.
Memory delta	Small payloads show tiny or noisy deltas; 1 MB shows the largest positive delta.	Process-level memory profiling is a coarse diagnostic and not an exact cipher memory-footprint measurement.
Stability plots	10 KB timing distributions are concentrated with occasional spikes.	Repeated runs are necessary. Single-run timing is not reliable for publication-quality benchmarking.
Security claims	No full-key exhaustive search, leakage trace, or attack success metric is present.	The study supports performance benchmarking.

4.1. Positioning against Reported Results

A direct numerical comparison between the runtimes reported here and results from published data is not considered. Reported AES throughputs in the constrained computing literature span several orders of magnitude for the same nominal algorithm, because they are governed by the implementation language, the availability of AES-NI or equivalent hardware instructions, the block cipher mode and the buffering strategy, rather than by the cipher specification. Placing an interpreted Python measurement beside a figure obtained from an assembly-optimised or hardware-accelerated implementation would compare two runtimes rather than two algorithms, and any conclusion drawn from such a comparison would be an artefact of the platforms involved.

The improvement offered by this study is therefore methodological, and it rests on three properties that the reviewed literature does not combine. Firstly, because every condition is executed 250 times, the standard deviations and the distributions in Figures 5 and 7 make it possible to separate a reproducible performance characteristic from a single anomalous execution, which a single run measurement cannot do. The occasional spikes visible in Figures 6 and 8 would each have been reported as the result had one run been used.

Secondly, because the key, the plaintext and the host are held constant across all conditions, the difference in mean runtime between the 10 KB and 1 MB workloads is attributable to payload scaling, while the increased dispersion at 1 MB indicates environmental effects on the measurement.

Thirdly, because the security claim boundary is stated explicitly in Section 3.2 and summarised in Table 3, the reduced key demonstration cannot be mistaken for an attack on a full AES configuration. The contribution is thus a result that is reproducible and whose limits are declared, and an

independent group following the stated protocol on comparable hardware should recover the same qualitative behaviour.

5. Conclusion

The correctness of the AES-128 implementation was assessed against a standard cryptographic test vector, followed by a round-trip encryption and decryption check. Four representative payload sizes were assessed. 250 repeated benchmark executions were performed to analyse timing and memory behaviour through graphical visualisation, and a histogram-based analysis along with a time-series examination were utilised as run-time stability checks.

The experimental results show that while the evaluated Python-based implementation shows largely consistent behaviour for small and medium payloads, under the 1 MB workload condition it experiences severe runtime growth and considerable execution dispersion. As the payload increases, memory-delta measurements show an increase. For small workloads, the measurements were noisy. Python's own memory take time dominates overall performance relative to CPU time for context switches, between processes and other operating-system effects. All of these observations combine to support the need for repeated-run benchmarking, deployment-specific performance evaluation, statistical variance reporting and caution when interpreting implementation-level cryptographic measurements. Future work can follow the proposed benchmarking framework towards strongly controlled multi-cipher comparison, embedded-device energy profiling, RAM and code-size evaluation, authenticated-encryption benchmarking and experimentally validated side-channel leakage analysis.

References

[1] Mohammed N. Alenezi et al., "On the Performance of AES Algorithm Variants," *International Journal of Information and Computer Security*, vol. 23, no. 3, pp. 322-337, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[2] Galih Putra Riatma et al., "Performance Evaluation and the Impact of File Size on Various AES Encryption Modes," *JARTEL JOURNAL: Telecommunication Network Journal*, vol. 15, no. 2, pp. 121-128, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[3] Sumit Singh Dhanda, Brahmjit Singh, and Poonam Jindal, "Lightweight Cryptography: A Solution to Secure IoT," *Wireless Personal Communications*, vol. 112, no. 3, pp. 1947-1980, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[4] Muhammad Rana, Quazi Mamun, and Rafiqul Islam, "Lightweight Cryptography in IoT Networks: A Survey," *Future Generation Computer Systems*, vol. 129, pp. 77-89, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[5] Monika Jangra, and Buddha Singh, "Performance Evaluation of SIMON and SPECK Block Ciphers to Secure IoT-Enabled Smart Cities," *Advanced Computing and Intelligent Technologies, Lecture Notes in Electrical Engineering*, Springer, Singapore, vol. 914, pp. 451-461, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[6] Abeer Tariq Maolood, Ekhlas Khalaf Gbashi, and Eman Shakir Mahmood, "Novel Lightweight Video Encryption Method based on ChaCha20 Stream Cipher and Hybrid Chaotic Map," *International Journal of Electrical and Computer Engineering*, vol. 12, no. 5, pp. 4988-5000, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Indu Radhakrishnan, Shruti Jadon, and Prasad B. Honnavalli, "Efficiency and Security Evaluation of Lightweight Cryptographic Algorithms for Resource-Constrained IoT Devices," *Sensors*, vol. 24, no. 12, pp. 1-19, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Mohammed El-hajj, Hussien Mousawi, and Ahmad Fadlallah, "Analysis of Lightweight Cryptographic Algorithms on IoT Hardware Platform," *Future Internet*, vol. 15, no. 2, pp. 1-29, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[9] Catarina Silva et al., "Analysis of the Cryptographic Algorithms in IoT Communications," *Information Systems Frontiers*, vol. 26, no. 4, pp. 1243-1260, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[10] Tiberius-George Sorescu et al., "Comparative Performance Analysis of Lightweight Cryptographic Algorithms on Resource-Constrained IoT Platforms," *Sensors*, vol. 25, no. 18, pp. 1-17, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[11] Ventura Sarasa Laborda et al., "Study about the Performance of ASCON in Arduino Devices," *Applied Sciences*, vol. 15, no. 7, pp. 1-34, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[12] A.V. Nageswara Rao et al., "End-to-End Encryption in IoT System with AES Technique," *Proceedings of the 1st International Conference on Research and Development in Information, Communication and Computing Technologies (ICRDICCT 2025)*, vol. 3, pp. 105-112, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[13] William J. Buchanan, Shancang Li, and Rameez Asif, "Lightweight Cryptography Methods," *Journal of Cyber Security Technology*, vol. 1, no. 3-4, pp. 187-201, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[14] Balajee Maram et al., "Lightweight Cryptography-based Deep Learning Techniques for Securing IoT-based E-Healthcare System," *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, Pudukkottai, India, pp. 1334-1341, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[15] Abel Yeboah-Ofori et al., "Applied Cryptography in Network Systems Security for Cyberattack Prevention," *2021 International Conference on Cyber Security and Internet of Things (ICSIoT)*, France, pp. 43-48, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[16] Abubakar Wakili, and Sara Bakkali, "Privacy-Preserving Security of IoT Networks: A Comparative Analysis of Methods and Applications," *Cyber Security and Applications*, vol. 3, pp. 1-15, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[17] Paul C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and other Systems," *Advances in Cryptology - CRYPTO '96, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, vol. 1109, pp. 104-113, 1996. [CrossRef] [Google Scholar] [Publisher Link]

[18] Saad Khan et al., "A Comprehensive Review on Lightweight Cryptographic Mechanisms for Industrial Internet of Things Systems," *ACM Computing Surveys*, vol. 58, no. 1, pp. 1-37, 2025. [CrossRef] [Google Scholar] [Publisher Link]