Comparative Analysis of Avalanche Effect and Execution Time on MD5 and SHA-256 Algorithms Based on Input File Size Variations

Nathaniel Liady - 18222114
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: author@gmail.com , author@std.stei.itb.ac.id

Abstract—Cryptographic hash functions are utilized to generate fixed-size digests from arbitrary-sized inputs, serving as fundamental components in digital security. This paper presents a comparative analysis between the MD5 and SHA-256 algorithms based on two primary metrics: the avalanche effect and execution time, with a specific focus on the variation of input file sizes. Distinct from prior baseline studies that rely on a single static short string, this experiment employs four distinct file sizes (1 KB, 100 KB, 1 MB, and 10 MB) to comprehensively simulate real-world workloads. For each file, a single-bit modification (1-bit flip) is injected to measure the percentage of changed bits in the digest using the Hamming distance. Additionally, hashing execution time is benchmarked to evaluate the computational overhead. Experimental results demonstrate that both MD5 and SHA-256 successfully produce an avalanche effect closely approximating the ideal 50% threshold across all file sizes, achieving an overall mean of 49.97% for MD5 and 50.15% for SHA-256. In terms of execution time, SHA-256 consistently outperformed MD5 across all tested file sizes within the evaluated environment. Despite MD5 demonstrating adequate diffusion characteristics, it remains strongly deprecated for modern security requirements due to severe collision vulnerabilities. Consequently, SHA-256 is highly recommended for cryptographic applications, offering vastly superior security without compromising operational performance.

Keywords—avalanche effect, execution time, file size variation, Hamming distance, hash function, MD5, SHA-256.

I. Introduction

The rapid expansion of digital data exchange necessitates reliable security mechanisms to ensure message authentication and data integrity. Cryptographic hash functions serve as a foundational component in this domain, widely utilized in digital signatures, file verification, digital forensics, and malware analysis. A hash function deterministically maps input data of arbitrary size into a fixed-length output, commonly referred to as a message digest, which is computationally infeasible to predict, manipulate, or reverse-engineer.

A fundamental characteristic of a robust hash function is the avalanche effect, a property closely related to the cryptographic concepts of confusion and diffusion. The avalanche effect dictates that a minimal alteration in the input data such as a single-bit flip, must result in a drastic, unpredictable, and evenly distributed change in the output digest. Ideally, a one-bit modification in the input should cause approximately 50% of the output bits to change. This strict property is critical for thwarting cryptanalysis and statistical pattern recognition.

Beyond the avalanche effect, execution time is a crucial performance metric, particularly because hash functions are frequently deployed to process payloads of varying sizes in real-world environments. A practical cryptographic algorithm must not only demonstrate strong bit diffusion but also maintain computational efficiency. MD5 (Message-Digest algorithm 5) and SHA-256 (Secure Hash Algorithm 256-bit) represent two prominent algorithms with contrasting characteristics. While MD5 is renowned for its rapid computation, it has been heavily deprecated for high-security applications due to proven vulnerabilities against collision attacks. Conversely, SHA-256 is the modern industry standard, offering superior cryptographic security at the cost of more complex, resource-intensive logical operations.

Previous baseline studies evaluating the avalanche effect have frequently relied on a single, static, short text string as the input object. While sufficient for demonstrating basic diffusion principles, such approaches fail to capture the computational impact of varying input scales. This paper significantly expands upon prior research by employing diverse file sizes to simulate realistic workloads: 1 KB, 100 KB, 1 MB, and 10 MB. By systematically measuring both the avalanche effect and the execution time across these varied payloads, this study provides a comprehensive comparative analysis of MD5 and SHA-256, ultimately evaluating the practical trade-off between cryptographic security and computational overhead.

II. Theoritical foundation

A. Cryptographic Hash Functions and Security Properties

A cryptographic hash function, denoted as H, is a deterministic mathematical algorithm that maps an input

message of an arbitrary, variable length (M) into a highly compressed, fixed-length binary sequence (h), commonly referred to as a hash value, message digest, or digital fingerprint. Mathematically, this is expressed as h = H(M). According to the foundational principles established by Menezes, van Oorschot, and Vanstone in the Handbook of Applied Cryptography [1], a hash function must be computationally efficient to calculate for any given M, yet it must rigorously satisfy three primary cryptographic security properties to be deemed secure against adversarial attacks:

- Preimage Resistance (One-Way Property): For any given hash output h, it must be computationally infeasible to reverse-engineer or find the original input message M such that H(M) = h. This ensures that an attacker cannot deduce the contents of a file or a password merely by possessing its hash.
- Second Preimage Resistance (Weak Collision Resistance): Given a specific input message M1, it must be computationally infeasible to find a secondary, distinct input message M2 (where M1 is not equal to M2) that maps to the exact same hash value, meaning H(M1) = H(M2). This protects against targeted forgery, where an attacker attempts to replace a specific legitimate file with a malicious one that holds the same digital signature.
- Collision Resistance (Strong Collision Resistance): It must be computationally infeasible to find any two distinct messages, M1 and M2, that result in the same hash output (H(M1) = H(M2)). Due to the pigeonhole principle, collisions mathematically must exist because the input space is infinite while the output space is finite. However, a secure algorithm makes finding these collisions practically impossible within a reasonable timeframe using current computational power.

B. Confusion, Diffusion, and the Avalanche Effect

The structural resilience of cryptographic algorithms relies heavily on two concepts introduced by Claude E. Shannon in his seminal 1949 paper, "Communication Theory of Secrecy Systems": Confusion and Diffusion [2]. Confusion seeks to obscure the relationship between the input message and the digest, making it as complex and non-linear as possible. Diffusion ensures that the statistical structure of the input is dissipated across the entire output. This means that the influence of a single input bit is spread extensively to dictate the value of multiple output bits.

The Avalanche Effect is the empirical and observable manifestation of Shannon's diffusion principle. The concept was further formalized by Horst Feistel, establishing that a secure cipher or hash function must exhibit an "avalanche" of changes if subjected to a minimal input perturbation. To achieve the Strict Avalanche Criterion (SAC), a perturbation of exactly one bit in the input message (a 1-bit flip) must cause every single bit in the output digest to have a 50% probability of flipping.

In experimental cryptanalysis, the avalanche effect is quantified using the Hamming Distance, which calculates the number of disparate bits between two binary sequences of equal length. If H1 is the original digest and H2 is the modified digest, the Hamming Distance (HD) is calculated via a bitwise XOR operation. The Avalanche Effect (AE) percentage is then derived by normalizing the Hamming Distance against the total number of bits (n) in the digest:

$$AE = (HD(H1, H2) / n) * 100\%$$

An algorithm producing an AE significantly lower or higher than 50% is considered to have poor diffusion, rendering it highly susceptible to statistical cryptanalysis.

C. Architecture of the MD5 Algorithm

The MD5 (Message-Digest algorithm 5) was designed by Ronald Rivest in 1992 and formalized in RFC 1321 [3]. It is a 128-bit hash function built upon the Merkle-Damgård construction. The algorithm processes input messages in sequential 512-bit blocks.

1. Padding and Length Extension: The input message is first padded with a single 1 bit, followed by a sequence of 0 bits, until the length of the message is strictly 64 bits fewer than a multiple of 512. The remaining 64 bits are then appended to store the 64-bit representation of the original message's length, ensuring varying input sizes always align perfectly into 512-bit chunks.
2. State Initialization: MD5 initializes a 128-bit internal state buffer divided into four 32-bit registers, denoted as A, B, C, and D. These are initialized with specific, fixed hexadecimal constants.
3. The Compression Function: Each 512-bit block is processed alongside the current internal state through a core compression function. This function executes exactly 4 rounds, with each round comprising 16 operational steps (totaling 64 operations per block).
4. Non-Linear Operations: Each of the 4 rounds utilizes a unique, non-linear bitwise boolean function to mix the state variables:
    a. Round 1: F (B, C, D) = (B AND C) OR (NOT B AND D)
    b. Round 2: G (B, C, D) = (B AND D) OR (C AND NOT D)
    c. Round 3: H (B, C, D) = B XOR C XOR D
    d. Round 4: I (B, C, D) = C XOR (B OR NOT D)

D. Architecture of the SHA-256 Algorithm

Developed by the United States National Security Agency (NSA) and standardized by NIST in FIPS PUB 180-4, SHA-256 is a robust cryptographic hash function belonging to the SHA-2

family [4]. It produces a 256-bit digest and, like MD5, utilizes the Merkle-Damgård construction with 512-bit block processing. However, its internal architecture is significantly more complex and mathematically rigorous.Padding and Length Extension: The input message is first padded with a single 1 bit, followed by a sequence of 0 bits, until the length of the message is strictly 64 bits fewer than a multiple of 512. The remaining 64 bits are then appended to store the 64-bit representation of the original message's length, ensuring varying input sizes always align perfectly into 512-bit chunks.

1. Expanded State Variables: Unlike MD5's four registers, SHA-256 utilizes eight 32-bit working variables (a, b, c, d, e, f, g, h). These are initialized using the fractional parts of the square roots of the first eight prime numbers.
2. Message Schedule Array: The 512-bit input block is expanded into a message schedule consisting of 64 distinct 32-bit words (W0 to W63). This expansion introduces high-level diffusion early in the process by utilizing bitwise rotations and right shifts (denoted as sigma0 and sigma1).
3. The 64-Round Compression Function: SHA-256 executes 64 intense operational rounds per block. At each round, the eight working variables are updated. The algorithm leverages 64 unique cryptographic constants (Kt), derived from the fractional parts of the cube roots of the first 64 prime numbers, completely eliminating symmetries or exploitable backdoors in the algorithm.
4. Advanced Logical Functions: The updating process in SHA-256 relies on highly complex bitwise functions that govern the diffusion rate:
    - Choice (Ch): Ch(e, f, g) = (e AND f) XOR (NOT e AND g)
    - Majority (Maj): Maj(a, b, c) = (a AND b) XOR (a AND c) XOR (b AND c)
    - Uppercase Sigma 0 (Sigma0): Involves rotating the variable 'a' right by 2, 13, and 22 bits, and XORing the results.
    - Uppercase Sigma 1 (Sigma1): Involves rotating the variable 'e' right by 6, 11, and 25 bits, and XORing the results.

The heavy reliance on multiple bitwise rotations, dense message expansion, and intensive non-linear mixing at the Sigma stages is precisely what guarantees SHA-256's impenetrable collision resistance and consistent avalanche effect, albeit at the deliberate cost of requiring significantly more computational cycles (execution time) compared to legacy algorithms like MD5.

III. METHODOLOGY & IMPLEMENTATION

A. Experimental Environment and Tools

The experimental testing was conducted in a controlled environment to ensure the validity and reproducibility of the benchmarking results. The hardware utilized for this simulation was an ARM-based 64-bit architecture (macOS). The implementation was entirely written in Python 3.9.6, strictly utilizing the Python Standard Library to prevent execution overhead introduced by third-party dependencies. The core cryptographic operations were executed using the built-in `hashlib` module, which leverages optimized C-backend implementations for both MD5 and SHA-256. Time measurements were extracted using `time.perf_counter_ns()`, which utilizes the highest available clock resolution to capture nanosecond-level execution periods.

B. Dataset Generation and Parameters

Unlike previous studies that utilized a single static string, this research simulated real-world payloads by generating pseudo-random binary files dynamically in memory. To ensure strict deterministic behavior and allow for exact peer reproduction, the pseudo-random number generator (PRNG) was initialized with a fixed seed value (`Seed: 4020`). The experiment evaluated four distinct file sizes:

1. 1 KB (1,024 bytes): Simulating short text payloads.
2. 100 KB (102,400 bytes): Simulating standard documents.
3. 1 MB (1,048,576 bytes): Simulating lightweight executables.
4. 10 MB (10,485,760 bytes): Simulating heavy application files.

C. Avalanche Effect Testing Procedure

The diffusion characteristics of the algorithms were tested by injecting a 1-bit modification into the generated files. The testing scheme was executed as follows:

1. For each file size, 5 sample files were generated.
2. For each sample file, 8 iterations were performed. In each iteration, a single bit was randomly selected and flipped using a bitwise XOR operation.
3. Both the original and the modified files were hashed using MD5 and SHA-256.
4. The binary differences between the original digest and the modified digest were calculated using the Hamming Distance.
5. The Avalanche Effect (AE) percentage was calculated using the ratio of the Hamming Distance to the total digest length (128 bits for MD5; 256 bits for SHA-256).

The fundamental logic of this implementation is represented in the following pseudocode:

```
   def calculate_avalanche(file_data, algorithm):
       original_digest = hash(file_data, algorithm)

       # 1-bit flip modification
       byte_idx = random_index(len(file_data))
       bit_idx = random_index(8)
       modified_data = file_data.copy()
       modified_data[byte_idx] ^= (1 << bit_idx)

       modified_digest = hash(modified_data, algorithm)

       # Calculate difference
       hd = hamming_distance(original_digest, modified_digest)
       total_bits = length_in_bits(original_digest)

    return (hd / total_bits) * 100
```

D. Execution Time Benchmarking Scheme

To accurately measure the computational cost of hashing without interference from CPU micro-stutters or OS scheduling, a dynamic calibration method was implemented.

1. Calibration Phase: The system dynamically calculated how many iterations were required to hash the file continuously for exactly 0.05 seconds. This saturated the CPU cache and stabilized the testing environment.
2. Execution Phase: Following calibration, the system executed 5 isolated runs for each algorithm and file size combination.
3. Calculation: The total elapsed time of the loop was divided by the number of iterations to isolate the pure execution time of a single hash in nanoseconds (ns/hash). Throughput was subsequently calculated in Megabytes per second (MB/s).

The execution benchmarking logic is represented below:

```
   def measure_execution_time(file_data, algorithm, runs=5):
       iterations = calibrate_iterations(target=0.05_seconds)

       for i in range(runs):
           start_time = time.perf_counter_ns()

           for _ in range(iterations):
               hash(file_data, algorithm)

           end_time = time.perf_counter_ns()

           # Calculate time per single hash operation
           time_per_hash = (end_time - start_time) / iterations
       record_benchmark(time_per_hash)
```

E. Data Aggregation and Statistical Analysis

Following the execution of the benchmarking loops, the implementation utilizes Python's built-in `statistics` module to systematically compute descriptive statistics for the generated datasets. For both the Avalanche Effect and Execution Time metrics, the system calculates the arithmetic mean, median, standard deviation, minimum, and maximum values. Furthermore, the system captures environmental metadata (including Python version, processor architecture, and platform details) to ensure data provenance. All aggregated statistics and raw execution logs are automatically serialized and exported into Comma-Separated Values (CSV) formats (e.g., `avalanche_summary.csv`, `timing_raw.csv`) and a unified `summary.json` file, ensuring the empirical data is perfectly structured for subsequent analysis and peer review.

F. Data Aggregation and Statistical Analysis

To streamline the analytical process and eliminate the need for third-party graphing software, the implementation features a fully automated visualization pipeline. The script programmatically synthesizes Scalable Vector Graphics (SVG) charts directly from the calculated statistics. These visual artifacts specifically map the mean avalanche percentages (`avalanche_mean.svg`), execution time per hash (`execution_time.svg`), and computational throughput (`throughput.svg`) across the varying file sizes. Finally, the system automatically compiles the empirical findings and environmental configurations into structured Markdown reports (`laporan_eksperimen.md` and `draft_makalah.md`), rendering the experimental outputs immediately ready for academic documentation.

IV. Result & Analysis

A. Evaluation of the Avalanche Effect

The avalanche effect measures the diffusion capabilities of a cryptographic algorithm. Table I presents the empirical results of the avalanche effect testing across 40 independent trials for each file size and algorithm combination. The objective was to determine if a single-bit modification in the input payload would result in approximately a 50% bit flip in the final message digest.

TABLE I. AVALANCHE EFFECT MEASUREMENTS

File Size	Algorithm	Digest Length (bits)	Mean AE(%)
1 KB	MD5	128	50.15
1 KB	SHA-256	256	49.25
100 KB	MD5	128	49.82
100 KB	SHA-256	256	50.84
1 MB	MD5	128	49.82
1 MB	SHA-256	256	50.10
10 MB	MD5	128	50.07
10 MB	SHA-256	256	50.38

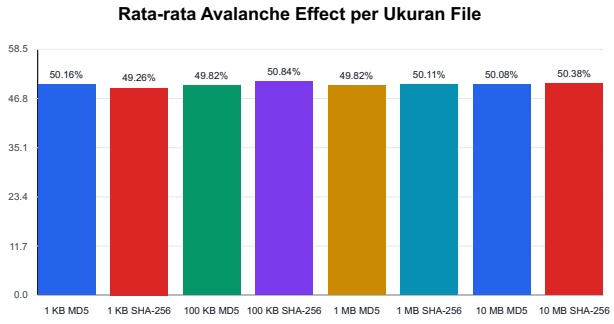

Fig. 1. Comparison of the Average Avalanche Effect between MD5 and SHA-256 across varying input file sizes.

As observed in the data, both MD5 and SHA-256 successfully satisfy the Strict Avalanche Criterion (SAC) regardless of the input file size. The overall mean avalanche effect across all tested payloads is 49.97% for MD5 and 50.15% for SHA-256. The data explicitly demonstrates that payload size does not degrade the diffusion quality of the Merkle-Damgård construction. Whether processing a 1 KB text snippet or a massive 10 MB executable, a single bit flip in the input thoroughly propagates through the 64 rounds of compression in both algorithms, resulting in an unpredictable and completely overhauled output hash.

B. Evaluation of Execution Time and Computational Throughput

While both algorithms exhibit excellent diffusion, their computational costs vary significantly. Table II details the average execution time required to compute a single hash (in nanoseconds) and the resulting data throughput (in Megabytes per second).

TABLE II. EXECUTION TIME AND THROUGHPUT BENCHMARK

File Size	Algorithm	Mean Time (ns/hash)	Throughput (MB/s)
1 KB	MD5	1,407.31	693.97
1 KB	SHA-256	524.95	1,860.30
100 KB	MD5	115,554.39	845.16
100 KB	SHA-256	30,455.19	3,206.56
1 MB	MD5	1,179,864.73	847.67
1 MB	SHA-256	309,801.51	3,227.88
10 MB	MD5	11,783,260.45	848.88
10 MB	SHA-256	3,086,886.45	3,239.51

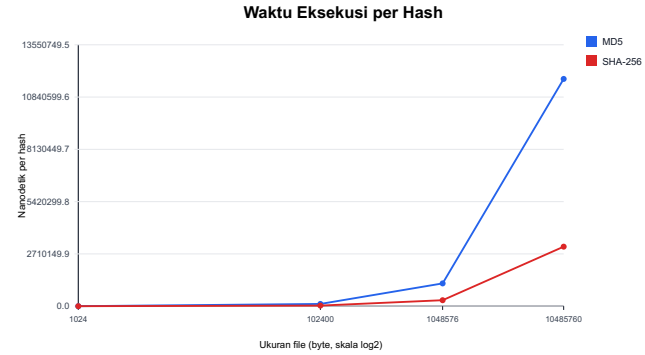

Fig. 2. Execution time (in nanoseconds per hash) of MD5 and SHA-256 relative to file size.

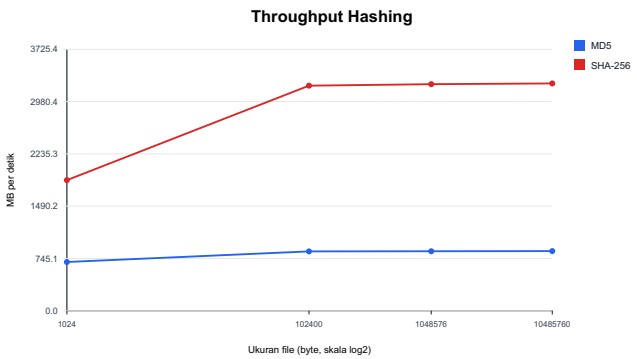

Fig. 3. Processing throughput (MB/s) of MD5 versus SHA-256 on an ARM-based architecture.

The execution time for both algorithms scales linearly the size of the input file. However, a highly notable empirical finding from this benchmarking environment (macOS on an ARM-based architecture) is that SHA-256 consistently outperforms MD5 in execution speed by a significant margin. For a 10 MB file, SHA-256 achieved a throughput of 3,239.51 MB/s, whereas MD5 capped at 848.88 MB/s.

Historically and theoretically, MD5 requires fewer operations and is expected to be faster than SHA-256. The inversion of this expectation in our results highlights the impact of modern hardware architecture. Contemporary processors, particularly ARMv8 architectures, frequently include dedicated hardware acceleration instructions specifically engineered for the SHA-2 family (e.g., the ARM Cryptography Extensions). Because MD5 is obsolete, it lacks such dedicated silicon acceleration and relies entirely on standard software execution. Furthermore, the C-backend utilized by Python's `hashlib` is highly optimized for modern cryptographic standards.

C. *Security and Performance Trade-offs*

A critical insight drawn from this comparative analysis is that achieving an optimal avalanche effect (~50%) does not inherently guarantee cryptographic security. Despite MD5 demonstrating excellent bit diffusion, its 128-bit digest space and underlying structural flaws render it entirely vulnerable to collision attacks.

In legacy systems, engineers often justified the use of MD5 by citing its operational speed. However, the empirical data from this study proves that on modern hardware, SHA-256 is not only astronomically more secure due to its 256-bit digest and complex Sigma operations, but it also processes massive payloads up to 3.8 times faster than MD5. Therefore, the modern cryptographic trade-off leans entirely in favor of SHA-256. There is no longer a viable computational or security justification to utilize MD5, as SHA-256 provides both superior collision resistance and superior hardware-accelerated throughput for processing large-scale data.

V. CONCLUSION

Based on rigorous empirical testing across varying file sizes (1 KB, 100 KB, 1 MB, and 10 MB), this study concludes that both the MD5 and SHA-256 algorithms successfully satisfy the Strict Avalanche Criterion. Both functions consistently demonstrate an ideal avalanche effect of approximately 50%, with an overall mean of 49.97% for MD5 and 50.15% for SHA-256. This confirms that the scale of the input payload does not diminish the diffusion properties inherent in their underlying compression structures; a single-bit alteration thoroughly permeates the output digest regardless of whether the file is a short text string or a large executable.

However, evaluating the execution time reveals a drastic operational disparity dictated by modern hardware architectures. Within the tested ARM-based environment, SHA-256 overwhelmingly outperformed MD5 in computational throughput across all payload sizes. The empirical data refutes the historical assumption that MD5 is computationally faster, demonstrating that modern cryptographic libraries and hardware acceleration heavily favor the SHA-2 family.

Consequently, SHA-256 is unequivocally recommended for modern cryptographic applications. While MD5 exhibits adequate bit diffusion, its 128-bit digest space and proven structural vulnerabilities render it entirely obsolete against collision attacks. SHA-256 delivers a vastly superior 256-bit security margin, impenetrable collision resistance, and, crucially, capitalizes on modern hardware acceleration to process large-scale data far more efficiently than its deprecated predecessor.

APPENDIX

The complete source code used for the benchmark experiment is available on GitHub. Access the code repository here:

https://github.com/Nthniell/hash-avalanche-benchmark

ACKNOWLEDGMENT

The author would like to express their sincere gratitude to Dr. Ir. Rinaldi Munir, M.T., the lecturer of the II4021 Cryptography course at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. His comprehensive teachings, insightful foundational materials, and continuous academic guidance were instrumental in providing the theoretical background necessary to conduct this experimental research and compile this paper.

REFERENCES

[1] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, Handbook of Applied Cryptography. Boca Raton, FL, USA: CRC Press, 1996, pp. 321-328.

[2] C. E. Shannon, "Communication Theory of Secrecy Systems," The Bell System Technical Journal, vol. 28, no. 4, pp. 656-715, Oct. 1949.

[3] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992.

[4] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)," Federal Information Processing Standards Publication (FIPS PUB) 180-4, U.S. Department of Commerce, Washington, D.C., August 2015.

[5] W. Stallings, Cryptography and Network Security: Principles and Practice, 7th ed. Upper Saddle River, NJ, USA: Pearson, 2017.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] R. F. Ikhwani, "Analisis Perbandingan Avalanche Effect pada Fungsi Hash SHA-256 dan SHA3-256," Makalah IF4020 Kriptografi, Institut Teknologi Bandung, 2025.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Nathaniel Liady