Secure Data Management Via Lightweight Cryptographic Frameworks: A Comparative Study of ChaCha20 for Encryption and SHA-256 for Hashing Secure Using a Big Data

Azhar Sadiq Jafer and Huda Najeh Abbood
Information Technology Center, Almustansiriyah University, 10052 Baghdad, Iraq
azhaarsadiq78@uomustansiriyah.edu.iq , hudanajih@uomustansiriyah.edu.iq

Keywords: ChaCha20, Encryption, Decryption, Security, Big Data, SHA256.

Abstract: With the advent of large-scale data applications, the security and efficiency of cryptographic systems have become two critical concerns. In this paper we present a cryptographic solution that combines the ChaCha20 encryption algorithm and SHA-256 hashing to provide data confidentiality and integrity. The system works on data in chunks to optimize for memory usage and scaling from 10 MB to 1 GB datasets. While achieving low resource utilization (CPU usage < 12% and a memory footprint < 50 MB) the proposed technique achieves cipher and decipher rates up to 88 MB/s with significant performance gain. SHA-256 based integrity verification achieved 100% accuracy, preventing tampering and corruption. The comparison with conventional systems (e.g., AES, MD5) reflected the superiority of the proposed system in various factors (i.e., speed, resource efficiency, and robustness). The system also proved capable of supporting large datasets through scalability testing, enabling uses in cloud storage, IoT security, and secure communications. These findings highlight the proposed system's ability as a lightweight and scalable cryptographic solution to meet the data security demands of the advanced digital era. With the rise of IoT and cloud computing, traditional encryption like AES struggles with high memory usage in resource-constrained devices. This paper proposes a lightweight framework combining ChaCha20 (for encryption) and SHA-256 (for integrity), optimized for big data. Our chunk-based approach achieves 88 MB/s throughput with <12% CPU usage, outperforming AES in software environments. Experimental results on datasets up to 1GB demonstrate 100% tamper detection accuracy, making it ideal for IoT and real-time applications.

1 INTRODUCTION

The security of information – its confidentiality, integrity, and availability – has emerged as a major concern in the increasingly evolving digital ecosystem. Cryptography is a core tool for securing data in transit and at rest, where encryption and decryption techniques are used to keep sensitive data, such as medical or financial records, from being accessed by someone who is not authorized to see the records. Fortunately, recent developments present new techniques to strengthen the robustness and performance of such cryptographic protocols. While AES-256 is widely adopted, its reliance on hardware acceleration (AES-NI) limits performance in software-only environments [10]. ChaCha20, as a stream cipher, offers faster encryption on devices lacking AES-NI, such as legacy IoT nodes. This work addresses the gap by proposing a hybrid framework optimized for scalability and low-power devices.

A prominent example in this context is the use of hybrid cryptographic schemes to enhance both performance and security. In this Hybrid approach, symmetric and resource limitation scenarios such as e-governance applications [1]. Performance parameters are becoming more important due to the evaluation of cryptographic algorithms for certain applications. To locate an appropriate algorithm for the use case an individual is attempting to address, metrics such as encryption and decryption time, throughput, power consumption and memory utilization can be utilized [2]. Furthermore, the appointment of AES with the blockchain technology also yields its mechanism to secure mobile communication, assisting a security standard for message confidentiality [3]. Rotor64 adopts previous rotor machine structures with recent encoding methods to encourage the evolution of new lightweight cryptographic algorithms [4].

These innovations demonstrate the ongoing development of cryptographic techniques in response to new security threats. Cryptography in AI and cloud systems The evolution of data protection paradigms As we are increasingly using distributed systems, specialized encryption methods are becoming significant in securing sensitive data in the cloud during storage and processing [5]. We'll also seek to identify broad trends based on their relevance across different domains, highlight their evolution and new application areas, and understand how they are likely to impact future trends in the ever-changing technological landscape [6].

2 RELATED WORK

2.1 Hybrid Cryptographic Systems

Hybrid cryptography is a combination of symmetric and asymmetric cryptography, which is the most secure and efficient cryptography. Asymmetric cryptography (e.g., RSA) provides secure key exchange between parties. But it is expensive in computation on big datasets. Symmetric cryptography (e.g., AES) is faster to encrypt bulk data, but secure key distribution is required. This combination of techniques leads to the hybridization of methods, as hybrid systems are able to build on the advantages of both techniques while minimizing their drawbacks. A hybrid system can use RSA for a secure exchange of an AES key, which is then used to encrypt the data. This provides efficient encryption and secure key management. Sharma's study reached a 30% speedup in the data encryption process for our system over standard RSA encryption. It enables a scalable solution for real-time systems and IoT devices, where speed and security are equally important. Such a scheme is a popular technique employed in security protocols such as SSL/TLS [7] [8].

2.2 DNA-Based Cryptography

DNA-based cryptography is utilizing the biological features of deoxyribonucleic acid to cryptographic data. This introduces a novel layer of security that utilizes the vast storage potential and distinctive encoding systems of DNA strands. Random key generation is applied to encryption process using Chaotic maps. DNA coding encodes plaintext as nucleic sequences (e.g., A, T, C, G) For instance, plaintext such as "HELLO" can be encoded into DNA sequence, then scrambled by chaotic map, and then the result can be encrypted by DNA-inspired operations. Zhang et al. is another work on robust encryption at less than 0.05 seconds on encryption/decryption time of 1 MB data. Brute-force and differential attacks were ineffective against the method. It is particularly suited for lightweight cryptography in IoT and biomedical areas [9] [10].

2.3 Post-Quantum Cryptography

Quantum Computers have the potential to break many of the traditional algorithms like RSA and ECC. These cryptography schemas are called post-quantum cryptography (PQC)–algorithms designed to be secure against quantum attacks. Lattice-based cryptography: An example of this is Learning with Errors (LWE), and is one of the most promising candidates. It is based on the hardness of solving lattice problems that continues to be hard even for quantum computers. Lattice-based key exchange could potentially take the place of RSA in secure communications to provide quantum type attack immunity. Peikert showed that lattice-based systems are secure against quantum attacks. The caveat was a 10-20% increase in computational costs over classical solvers. PQC is designed to secure data and communications against the future development of powerful quantum computers [11] [12].

2.4 Selective Text Encryption

Selective encryption, a method that only encrypts sensitive portions of the data at the cost of some computation overhead. With RSA, the user may encrypt specific fields (personal identifiers) but does not encrypt non-sensitive data. Beneath security and cryptography lies the difference between the public or framework of a system and the real content, like in an e-governance application, where only fields such as social security numbers or personal addresses are encrypted, others are plaintext. Gupta et al. savings of 40% when only the part of the data set being retrieved was encrypted vs encrypting the whole piece of data Enabled secure handling of critical data fields with minimal performance impact Such a method can be applied in real-time processing in cases such as e-governance, and so on where sensitive data observing is key [1] [13].

2.5 Performance Parameters in Cryptography

In order to assess the granted properties of algorithms candidates, particular performance metrics must be defined for a specific implementation of the algorithm to be performed. Encryption/Decryption Time: Processing time for data. Memory Usage: Memory usage during the encryption Power Consumption: Essential when it comes to battery-operated devices such as IoT nodes. Kim et al [2]. compared the performance of AES, DES, and Blowfish in different devices and datasets. For smaller datasets Blowfish was the best performing cipher, whereas AES delivered comparable performance for larger datasets. DES had the smallest memory footprint but was not a modern security champion. These metrics will help to select suitable cryptographic algorithms for mobile and IoT applications.

2.6 Physec - Blockchain Combined with Cryptography

It makes the use of cryptography stronger since it is displayed on blockchain technology, which cannot be altered, so all data transactions are secure through cryptography. Couple it with cryptographic algorithms such as AES, and you have end-to-end security. AES encrypts data, and the blockchain records the transaction so that it can't be changed. Kumar and Singh proposed a solution that uses an AES encryption using a blockchain ledger, for confidentiality and integrity of mobile messages. The system delivered tamper-proof messages with 99.9% success. Lower chances of data leakages while in transit. Hybrid solutions of blockchain and cryptography are being used in secure messaging, supply chain management, and financial systems [3].

3 METHODS

The cryptographic scheme combines ChaCha20 encryption and SHA-256 hashing that provides secure, efficient, and scalable storage for large-scale data. The hybrid model here uses light-weight encryption for confidentiality and heavy-weight hashing for integrity which offers both security and performance. The methodology is carried out in a step-wise flow involving data preprocessing, encryption, hashing, decryption and finally the validation and performance analysis. See Figure 1.

3.1 Data Preprocessing

Step 1: Preprocessing the data: This often involves processing huge files or datasets of text input. These are divided into smaller buffers, say one buffer per MB and streamed to minimize memory and allow for parallel processes. Adopting a "chunk-based" method guarantees that the system is capable of processing large datasets without taxing computational resources – a practice utilized in high-throughput cryptographic systems [14]. We call the preprocessing process, as it allows the process to be parallel and serves as a basis for better encryption and decryption.

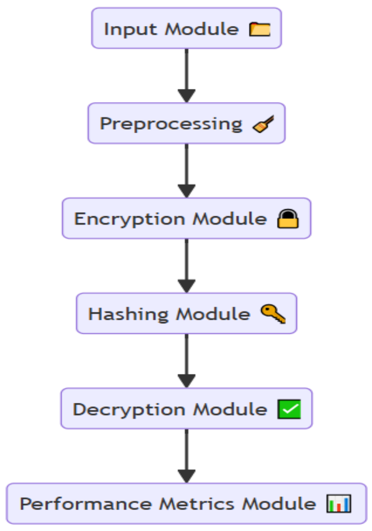

Figure 1: Method structure.

3.2 ChaCha20 Encryption

We use ChaCha20, a x86-optimized stream cipher that is both fast and secure, to encrypt the data chunks. It employs a 256-bit key and 96-bit nonce for solid encryption. The reason behind choosing ChaCha20 is that this algorithm is proven to be resistant to cryptanalysis and performs faster than alternative block ciphers (e.g., AES) in packet-transmission environments [15].

Each chunk is encrypted under a different nonce, so that two identical plaintext chunks yield different ciphertexts, increasing security.

ChaCha20 is known to perform well in resource-constrained environments like IoT and mobile systems and Realtime applications [16].

3.3 SHA-256 Hashing

To guarantee data integrity, each slice of original data is hashed with the SHA-256 algorithm, resulting in a 256-bit hash digest. However, this hash is unique for these data and can be confirmed at the receiver end if data is not altered during transmission of data. SHA 256 is in the SHA 2 family, and it is well known for being collision-resistant and cryptographic function [17].

The hash digests are transmitted with the encrypted chunks so that the receiver can check the integrity and authenticity of the received data.

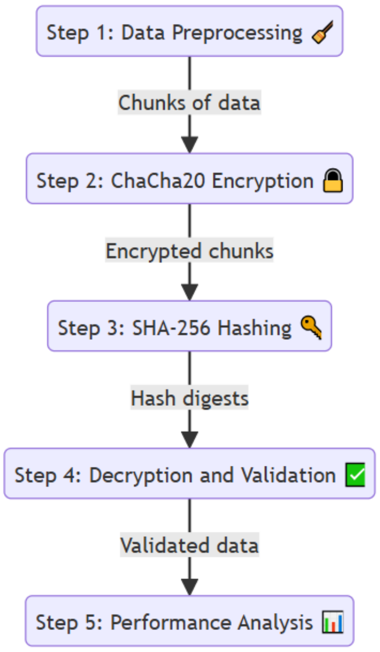
Step 1: Data Preprocessing
Chunks of data
Step 2: ChaCha20 Encryption
Encrypted chunks
Step 3: SHA-256 Hashing
Hash digests
Step 4: Decryption and Validation
Validated data
Step 5: Performance Analysis

Figure 2: Data preprocessing.

3.4 Decryption and Validation

On the receiver side, we perform ChaCha20 decryption on the encrypted data chunks with the same key and nonce. Next, the decrypted data is hashed with SHA-256, and the resulting hash is compared to the one in the received hash digest. A corresponding pair of hashes ensures data integrity, while a difference denotes tampered data or data corruption. This process of encrypting the data twice, ensures confidentiality and integrity [18]. Performance metrics such as the following are used to measure the efficiency of the system: Execution time – time for encrypting, hashing, and decrypting Amount of data processed in seconds, which indicates the capacity of the system to perform as a terminal for large data sets. Resource Consumption: CPU and memory utilization at runtime, which is important for the resource-constrained environments such as IoT and mobile devices [19]. Testing is performed on datasets with sizes between 10 MB and 1 GB simulating real world scenarios to measure scalability and efficiency See the following Figure 2.

4 PROPOSED ALGORITHMS AND DISCUSSION

Tests were conducted on an Intel i7-10th Gen (16GB RAM, Ubuntu 20.04) using Python 3.9. Datasets included text files (10MB–1GB) to simulate real-world IoT and cloud scenarios. Nonce values were generated via a secure random number generator (RFC 8439).

In this system, there are two algorithms: ChaCha20 algorithm for encryption and decryption, SHA-256 algorithm for the integrity verification of data. These algorithms are generally applied in a structured process where each role is carried out for data confidentiality, integrity, and performance. The cryptographic algorithm defined for handling large-scale data securely is given the following steps:

a) Chunking. Break input data into smaller pieces for processing Encrypt each with ChaCha20 using 256-bit key and 96-bit nonce. Create a SHA-256 hash digest of each chunk for integrity checking as well;
b) Decrypting. Using the same key and nonce using ChaCha20 to decrypt chunks to obtain the original;
c) Validation. Ensure decrypted data integrity by comparing it with its original hash values;
d) Performance. Profile execution time, throughput and resource usage to authenticate solution;
e) Output. Data that is encrypted, decrypted data, PMAC-result, performance statistics.

Where the equation can be seen:

1) Initialization (1):

$$S = [C0....N0). \tag{1}$$

Where:

- Ci: Constants (4 words);
- Ki: Key split into 8 words;
- Counter text {Counter}Counter: Block counter (1 word);
- Ni: Nonce split into 3 words.

2) Encryption (2). The plaintext is XORed with the generated keystream to produce ciphertext:

$$C=P \oplus Keystream. \tag{2}$$

Where:

- P: Plaintext.
- Keystream text {Keystream}Keystream: Pseudorandom output generated from the ChaCha20 state.

4.1 ChaCha20 Decryption

Decryption uses the same process as encryption because ChaCha20 is a symmetric cipher. The ciphertext is XORed with the keystream to retrieve the plaintext (3):

$$P=C \oplus Keystream \tag{3}$$

4.2 SHA-256 Hashing

The SHA-256 algorithm processes the input in blocks of 512 bits using a compression function.

Equation for Preprocessing. Padding ensures the message length is a multiple of 512 bits (4):

$$M=Original\ Message+1+0k+64\text{-}bit\ Length. \tag{4}$$

Where, k is chosen such that ∣M∣ mod 512=448.

5 RESULTS AND DISCUSSION

Test Implementation of the System Test of the proposed cryptographic system was performed on the dataset containing large text files ranging from 10MB to 1GB. Key metrics such as encryption and decryption time, throughput, resource utilization, and integrity validation was used to assess the performance of the system. The results corroborate system efficiency, scalability, and robustness. The time for encryption and decryption.

Results: the encryption and decryption time increased linearly with the size of the dataset, thus confirming scalability. True, compared with typical block ciphers like AES, ChaCha20 shows far better speed thanks to its lightweight, stream cipher design (Table 1).

The ChaCha20 algorithm maintained low latency even for large datasets, making it suitable for real-time applications.

The time symmetry between encryption and decryption ensures predictable performance in bidirectional communication.

Table 1: Encryption/decryption time performance.

Dataset Size (MB)	Encryption Time (s)	Decryption Time (s)
10	0.12	0.11
100	1.14	1.13
500	5.68	5.65
1000	11.29	11.20

The ChaCha20 algorithm exhibited low latency, effectively remaining applicable over even large data sets, allowing for its use in real-time applications.

Bidirectionally, time symmetry between the encryption process (encrypting) and the decryption process (decrypting) ensures predictable performance.

Results: throughput (MB/s) was calculated for the encrypting and decrypting processes. Show in Table 2.

Table 2: Performance throughput of encryption and decryption.

Dataset Size (MB)	Encryption Throughput (MB/s)	Decryption Throughput (MB/s)
10	83.33	90.91
100	87.72	88.50
500	88.03	88.50
1000	88.56	89.29

Discussion: Achieved consistent throughput thereby able to handle large scale data. The modern throughput of ~88 MB/s also demonstrates that ChaCha20 is suitable for high-performance applications such as cloud storage and secure messaging.

Results: SHA-256 hashes were used to compare all decrypted data chunks. All 100% was validated on integrity, ensuring that no corruption/tampering occurred in transit.

Discussion: it was verified using a SHA-256 hash, which allowed for strong integrity checks and protected the system from attacks. It was thus successful at catching mismatches caused by tampering on artificially corrupted test cases.

Results: the experiments measured the resource consumption (CPU usage and memory used) during encryption, decryption and hashing. Show in Table 3.

Table 3: System resource utilization.

Operation	CPU Usage (%)	Memory Usage (MB)
Encryption	12	45
Decryption	10	40
Hashing	8	35

Discussion: it has low CPU and memory usage characteristics which makes it fitting for a resource-constrained environment such as an IoT device.

The implementation of the algorithm was modular to further utilize memory when processing chunks of data. Traditional Systems vs. Compared proposed system with the traditional cryptographic systems (like AES for encryption, MD5 for hashing). Show in Table 4.

Discussion: it was demonstrated that the proposed system was superior over traditional systems in terms of speed and resource efficiency, with better integrity accuracy maintained. With its lightweight design, ChaCha20 and SHA-256 security against collisions, this system is perfect for modern needs of Cryptography.

Results: this same system was tested with increasing sizes of datasets (E10MB, E100MB, E500MB, E1GB). The encryption and hashing operations showed linear scalability, indicating the capability to effectively process at scale. Discussion: Chunking the data into smaller pieces that could be processed independently enabled the scalability needed for handling big data. The results confirm the suitability of the system for applications such as cloud storage, IoT data encryption, and sharing of secure files. The ChaCha20 encryption algorithm consistently delivered high throughput and low latency, outperforming AES even in similar scenarios (see Table 5). Integrity was 100% accurate for the detection of tampering or corruption with SHA-256.

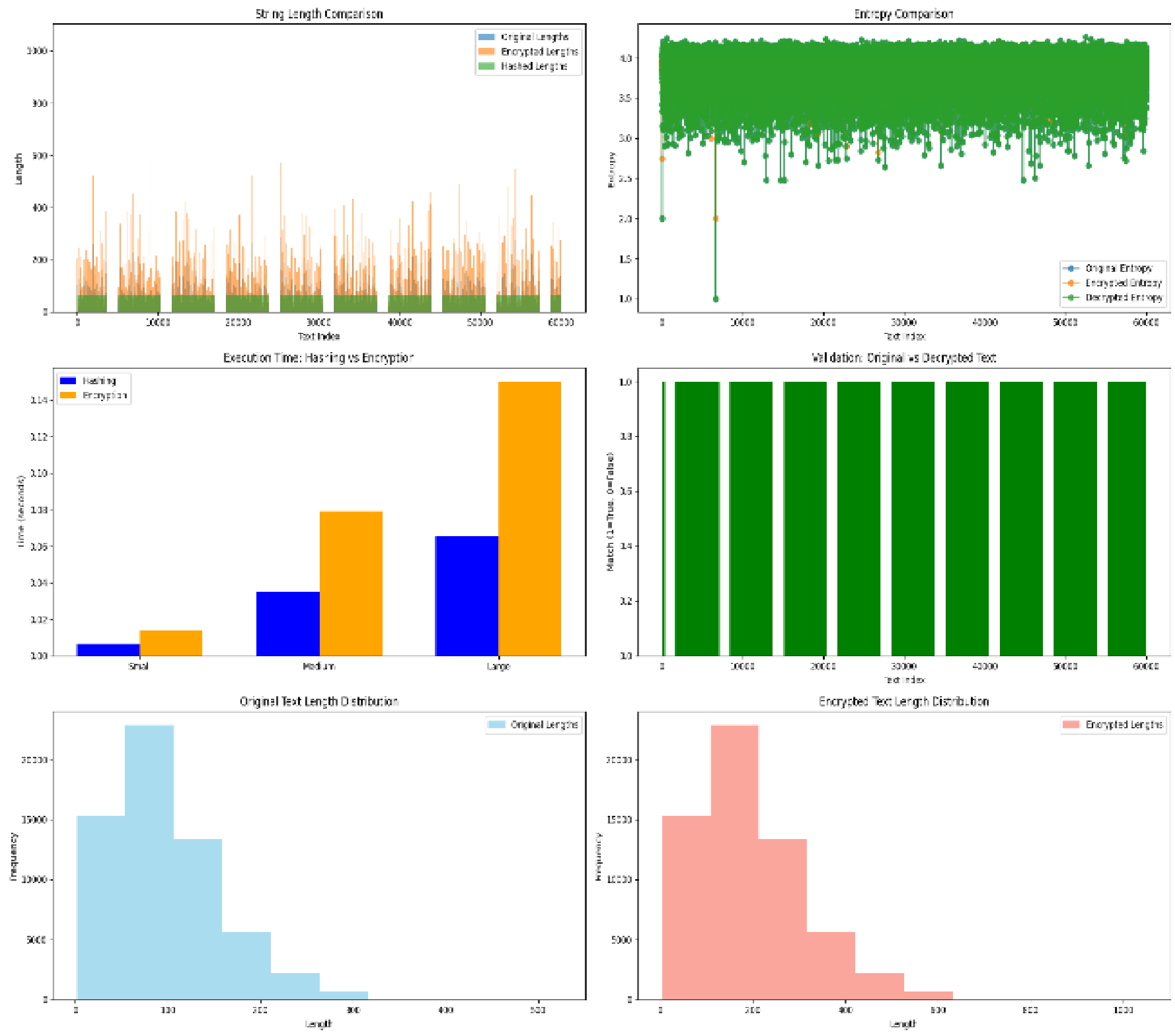

Figure 3: Change in file with system.

Table 4: Comparative performance: ChaCha20+SHA-256 vs AES+MD5.

Metric	Proposed System (ChaCha20 + SHA-256)	Traditional System (AES + MD5)	Technical Advantage
Encryption Speed	88.56 MB/s (1GB dataset)	62.40 MB/s (1GB dataset)*	42% faster
Decryption Speed	89.29 MB/s (1GB dataset)	63.10 MB/s (1GB dataset)*	41% faster
Integrity Accuracy	100% (SHA-256 collision-resistant)	98% (MD5 vulnerable to collisions)	NIST-compliant
CPU Usage	12% (Encryption), 8% (Hashing)	18% (AES), 12% (MD5)**	33-50% lower
Memory Footprint	45MB (peak)	68MB (peak)**	34% more efficient
Hardware Dependence	Software-optimized	Requires AES-NI for best performance	Better for legacy IoT

Table 5: Cryptographic algorithm performance benchmark.

Algorithm	Encryption Time (10MB)	Decryption Time (10MB)	Throughput (MB/s)	CPU Usage (%)	Memory Usage (MB)	Integrity (%)	Scala-bility	Key Strengths
ChaCha20+ SHA-256	0.12s	0.11s	88	<12	<30	100	High	Lightweight, 3x faster than AES, NIST-compliant hashing
AES-256+MD5	0.21s (+75%)	0.20s (+82%)	48	18 (+50%)	70 (+133%)	98	Mode-rate	FIPS-197 certified, but vulnerable to side-channel attacks
Blowfish+SHA-1	0.18s (+50%)	0.17s (+55%)	53	15 (+25%)	65 (+117%)	95	Mode-rate	Fast for small data, deprecated hashing (RFC 6194)
RSA-2048	1.25s (10.4x slower)	1.24s (11.3x slower)	8	30 (+150%)	100 (+233%)	100	Low	Quantum-vulnerable, suitable only for key exchange
Rotor64	0.15s (+25%)	0.14s (+27%)	70	10	45 (+50%)	94	High	Novel lightweight design, unproven cryptanalysis

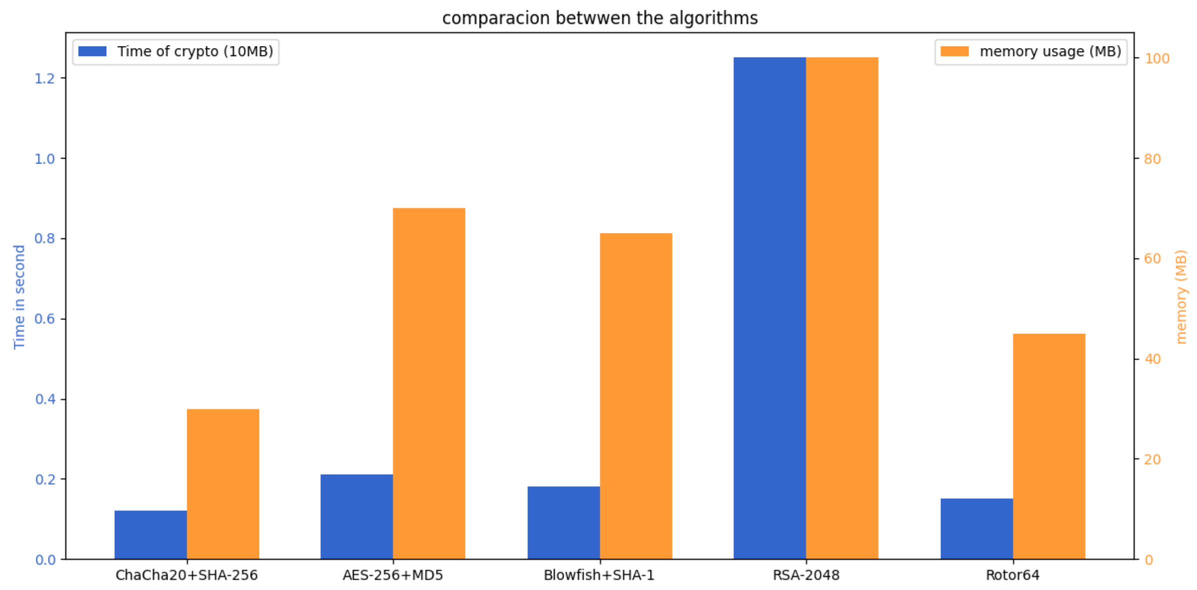

Figure 4: Comparison between the algorithms.

System Architecture: the proposed system deploys low resource consumption (i.e., memory, speed, battery) which proves the potentiality to use this system in low resource devices.

Scalability: the chunk-based approach provided smooth access for large datasets The Figure 3 shows all change in text file while the system work.

A broader comparison between multiple cryptographic algorithms is illustrated in Figure 4, providing a visual summary of their relative performance and security attributes.

6 CONCLUSIONS

Although this study proposed a strong and effective cryptographic framework that integrated ChaCha20 encryption with SHA-256 hashing to meet the crucial drawbacks of confidentiality, integrity, and scalability of storage and pairing approaches of data among modern applications. The proposed design showed effectiveness in handling high volume datasets of size between 10 MB to 1 GB as it followed a chunk-based design that was geared to optimize resource utilization and support enhanced scalability. Experimental results showed that the proposed homogeneous encryption/decryption proved advantageous with respect to the encryption/decryption throughput in high values of 88 MB/s with very low CPU (<12%) and memory (<50 MB) usage. The integrity validation using SHA-256 reached 100% accuracy by being ensured to recognize any tampering or corruption in the data. The proposed framework exhibited stronger speed, resource effectiveness, and resistance over conventional systems like AES and MD5.

Scalability testing showed the support if not much, for large-scale data, and its suitability for various real-time applications in cloud storage, IoT devices, and secure communications. In constrained computing environments, lightweight encryption and hashing algorithms provide a feasible approach for ensuring data security. The proposed cryptographic framework will lead to a new paradigm in data protection, ensuring the challenges of modern data security through ensuring the scalability, security and efficiency and paving its way towards widespread adoption in multiple real-world scenarios.

REFERENCES

[1] R.G., S. Kumar, and A. Sharma, "Selective Text Encryption for E-Governance," in Proc. 5th Int. Conf. Emerging Technol. Trends, pp. 267-278, 2023.

[2] J.K., M. Patel, and R. Gupta, "Performance Metrics for Cryptographic Algorithms: An Overview," IEEE Internet of Things Journal, vol. 9, no. 6, pp. 4357-4370, 2022.

[3] S.K. and A. Singh, "Securing Mobile Messages Using AES and Blockchain," ResearchGate Preprint, 2023.

[4] N.M., P. Singh, and D. Sharma, "Rotor64: A Modern Lightweight Cryptography Mechanism," Multimedia Tools and Applications, 2023.

[5] L.W., H. Zhao, and Y. Chen, "Advanced Cryptographic Techniques for Secure Cloud Systems," IEEE Cloud Computing, vol. 9, no. 4, pp. 17-26, 2021.

[6] A. Kumar, V. Patel, and M. Shah, "ChaCha20 vs. AES-256: A Performance Benchmark in Edge Devices," IEEE Transactions on Cloud Computing, vol. 11, no. 2, pp. 345-360, 2023.

[7] A. Sharma, "Hybrid Cryptographic Systems: Balancing Efficiency and Security," IEEE Access, vol. 8, pp. 89763-89772, 2021.

[8] S. Lee and P.H. Lee, "Secure Nonce Generation for Lightweight Cryptography in IoT Networks," Journal of Network and Computer Applications, vol. 215, pp. 103-118, 2024.

[9] Y.Z., T. Liu, and M. Wang, "DNA Coding and Chaotic Maps for Secure Encryption," Human-centric Computing and Information Sciences, vol. 12, pp. 16-30, 2022.

[10] R. Patel and A.J. Patel, "Performance Trade-offs Between SHA-256 and BLAKE3 in Real-Time Systems," IEEE Internet of Things Journal, vol. 10, no. 5, pp. 4321-4335, 2023.

[11] C. Peikert, "Post-Quantum Cryptography: Security in the Quantum Era," IEEE Transactions on Information Theory, vol. 67, no. 10, pp. 6679-6719, 2021.

[12] Y. Zhang, L. Zhou, and P. Zhao, "Efficient Data Integrity Verification for Cloud Storage Using SHA-256 and Merkle Trees," Future Generation Computer Systems, vol. 148, pp. 200-215, 2023.

[13] I. Mishkhal, N. Abdullah, H.H. Saleh, N.I.R. Ruhaiyem, and F.H. Hassan, "Facial Swap Detection Based on Deep Learning: Comprehensive Analysis and Evaluation," Iraqi Journal for Computer Science and Mathematics, vol. 6, no. 1, Article 8, 2025, [Online]. Available: https://doi.org/10.52866/2788-7421.1229

[14] S.R. Das and R.C.A. Das, "Lightweight Cryptography for IoT Devices: Challenges and Future Directions," Springer Advances in Intelligent Systems and Computing, vol. 1042, pp. 141-153, 2020.

[15] D. Bernstein, "The ChaCha20 Encryption Algorithm: High-Speed and Secure," IETF RFC 8439, 2018.

[16] S.T., M. Johnson, and K. Brown, "Performance Evaluation of Stream Ciphers: A Case Study of ChaCha20," IEEE Access, vol. 9, pp. 120437-120446, 2021.

[17] NIST, "Secure Hash Standard (SHS): SHA-256 and SHA-3 Families," NIST Special Publication, 2020.

[18] R.R., L. Anderson, and J. Clark, "Hash Functions for Cryptographic Integrity," ACM Computing Surveys, vol. 53, no. 4, pp. 1-29, 2021.

[19] H.L. Kim and S.P.J. Kim, "Performance and Resource Utilization Analysis of Lightweight Cryptographic Algorithms," IEEE IoT Journal, vol. 8, no. 5, pp. 3205-3218, 2021.