International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 – 5161

Available at http://inpressco.com/category/ijcet

Research Article

A Comprehensive Performance Study of Hashing Algorithms: MD5, SHA-256, and SHA-512 using Java

Mahathi Kari*

Independent Researcher

Received 01 Dec 2023, Accepted 24 Dec 2023, Available online 26 Dec 2023, **Vol.13, No.6 (Nov/Dec 2023)**

Abstract

The algorithms of hashing are the key to information protection in the modern world of digitality. They provide much-needed features such as integrity of and authentication of data, as well as secure data storage in digital and cloud-based data storage. In a very significant sense, MD5, SHA-256, and SHA-512 are the most important cryptographic hash functions that have stayed at the center of software applications globally, even though they vary in terms of security as well as performance. The Java Cryptography Architecture (JCA) and Java Cryptography Extension (JCE) are used in this study to begin a performance investigation of the three hashing methods. The analysis encompasses the length of hash, processing rate, padding and the complexity of computation. Even though MD5 is faster, it can be affected by collisions and brute-force attacks; therefore, it cannot be trusted to work in highly secured settings. SHA-256 and SHA-512 have a significantly stronger cryptographic security than MD5, and SHA-512 has the greatest security against attacks (although it consumes more computing resources). The Java-based implementation reveals the trade-offs between performance and security which help developers to make the adequate choice of hashing algorithms. In addition, the paper suggests the need to use secure coding practices when using cryptographic APIs in Java.

Keywords: *Cryptographic Hash Functions, Java Cryptography Architecture, MD5, SHA-256, SHA-512, Data Integrity, Performance Analysis.*

Introduction

In modern cryptoanalysis, hash functions are a crucial component of cryptographic systems, which use digital signatures as one of their processes, Authentication schemes, cryptographic accumulators, and safer communication [1]. Cryptography provides mathematical means to ensure information confidentiality, integrity, and authentication, and cryptographic hash functions satisfy strict security properties [2]. Developing a cryptographic hash from variable-length data to a fixed-length digest is one step in the process. This ensures that any changes to the input significantly impact the outcome. Likelihood of a highly probable outcome.

As cloud computing becomes more popular, issues such as secure password storage and data integrity verification have become the main concerns. Never again passwords be stored in plain text; on the contrary, they concealed as hashes which are the output of a cryptographic hash function. However, using weak or old algorithms for cryptographic processing still endangers such systems [3].

*Corresponding author's ORCID ID: 0000-0000-0000-0000
DOI: https://doi.org/10.14741/ijcet/v.13.6.15

As an illustration, MD5 and SHA-1 are still used in a large number of cases whereas they are known to be vulnerable, in fact, among the vulnerabilities are collision attacks and brute-force exploits [4]. Web applications frequently use hashing for user authentication [5]Thus, the problem of choosing a safe algorithm is the key to security against unauthorized access if there is a database breach.

Development teams have access to modern encryption libraries such as Java Cryptography Architecture (JCA) and Java Secure Socket Extension (JSSE), which offer standardized APIs for secure hashing and communication protocols [6]. However, if these APIs are misused and the company lacks sufficient cybersecurity expertise, the resulting misconfigurations and vulnerabilities could be easily exploited. Besides, MD5, despite its historical role in digital forensics and data identification [7], is a security weakness that suffers from collisions, so it has been ruled out from the list of modern application encryption algorithms.

The National Institute of Standards and Technology (NIST) created the SHA-2 suite and then added SHA-3 to improve the security of the algorithms, as MD5 and SHA-1 were deemed insufficient in the realm of

cryptography [8]. Nevertheless, MD5, SHA-256, and SHA-512 continue to be the go-to for a majority of software systems, which is why the performance and suitability of those mentioned above over different platforms still require empirical testing.

This study compares the performance of MD5, SHA-512, and SHA-256 with the help of the Java programming language based on certain parameters like execution time, efficiency of digest generation, and amount of computation needed. The findings are intended to support developers and security practitioners in choosing the right hashing algorithms for today's software.

Organization of the Paper

The paper is organized as follows Section II gives a thorough explanation of the MD5, SHA-256, and SHA-512 algorithms and how they work. A thorough explanation of the Java implementation utilizing JCA and JCE is covered in Section III. The primary challenges of using hash algorithms with Java are covered in Section IV. A review of relevant literature on cryptographic hashing and security enhancements is conducted in Section V. The study's main conclusions and recommendations for additional research are presented in Section VI.

Sha-512 and Sha-256, Md5 Hashing Algorithms: an Overview

Ensuring data integrity and protection requires using high-grade cryptographic hash functions SHA-512. Among them, MD5 was the fastest, producing a 128-bit message digest; however, it is already considered unsafe due to its susceptibility to collision attacks. On the contrary, SHA256 and SHA512, both from the SHA-2 family, produce 256-bit and 512-bit hashes, respectively, providing much stronger protection against attacks than the previous ones [9]. SHA-256 is the usual choice for implementing blockchain technologies and digital signatures; on the other hand, SHA-512 is favoured in high-security settings with greater computational power. The said algorithms serve as the main components of current authentication and data security mechanisms. The structural, security, and performance elements of these algorithms are compared, and Table I displays the findings, which provide a thorough overview of the differences while highlighting key elements such as hash size, block size, padding, the number of processing rounds, and typical applications.

Message-Digest Algorithm 5 (MD5)

Messages of different sizes are converted into a 128-bit hash value using the MD5 procedure. Calculating the message from its digest is difficult since each message digest is unique [10]. It makes understanding the security and fundamental ideas of the MD5 algorithm particularly challenging. Improvements to the MD5 algorithm and its implementation in a security user application are suggested for brute-force and collision attacks. The user's security can be guaranteed by changing the plaintext and ciphertext structures. The first input is taken from the user in binary or text (String) format, and the MD5 function is then called with it. As shown in Figure 1, the MD5 function instantiates the Message Digest class by supplying the algorithm name. The method then returns an object of the Message Digest class that implements the algorithm, which is subsequently supplied as a parameter to the class. The update function is then called using the input array of bytes, the beginning point, and the input length to update the Message Digest object. The concluding phase, like padding, is then carried out to complete the hash calculation [11]. After that, the hash value is converted to a Big Integer by calling the Big Integer constructor with the bits and an array of bytes, then converted to a String, and finally returned to the function.

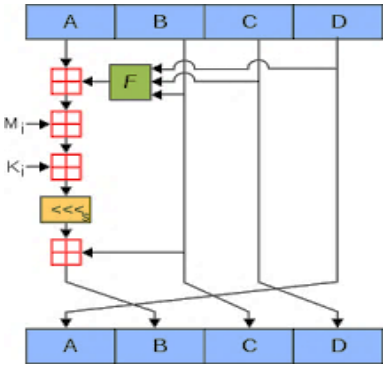

Fig.1 MD5 Algorithm Structure

Appending Padding Bits: Adding extra bits to the end of the original message is known as padding. The MD5 method requires that the data be analyzed in 512-bit message blocks [12]. Therefore, if the length of the original message is less than 512 bits, it is padded to the nearest multiple of 512, with a 64-bit padding, until its length equals 448 mod 512. After that, the pieces are added to the original plain text.

Dividing: The messages are split up into n 512-bit sub-blocks during this procedure, each of which is an exact multiple of 512.

Initialising MD Buffers: The length of each of the four initialized chaining variables—A, B, C, and D—is 32 bits, for a total of 128 bits. The input is processed, and the Message Digest (final output) is stored in these initialized chaining variables.

Processing: Pre-initialized MD buffers are used to process each divided 512-bit sub-block. The 512-bit sub-blocks should then be split into 16-32 blocks, with each block containing 32 bits.

SHA-256 Algorithm

On 32-bit computers, SHA-256 is quicker. On 64-bit computers, SHA-512 is quicker. Compared to SHA-256,

SHA-512 contains 25% extra rounds. SHA-256 uses 512 bits at once for 64 rounds of its compression operation. Five hundred twelve bits at a time are compressed in 80 rounds by SHA-512 [13]. SHA-512 is superior. For hashing more than 16 bytes of data at once, use SHA-256. The two steps of the SHA-256 computation are shown in Figure 2. The first step is pre-processing the initial communications. Message expansion and padding are required for the round computation [14]. Adding bits in accordance with predetermined guidelines until the overall length is an integer of 512 bits is known as padding. Each 512-bit block is then enlarged to 64x32 bits for the SHA-256 round calculation.

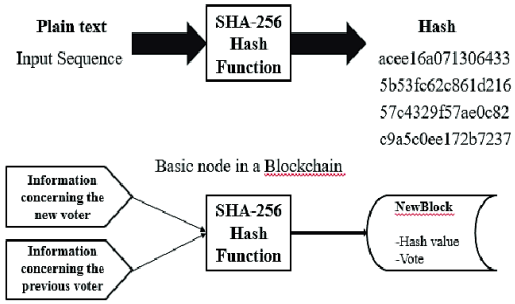

Fig.2 SHA 256 Algorithm

Message Pre-processing: It hashes the original message. The message's total length is padded to a multiple of 512 bits. Padding is added to split the message into the fixed-size blocks required by the SHA-256 algorithm. A single bit1, a series of 0 bits, and finally the message's length in 64-bit format are used for padding.

Message Expansion: Padding in the next step, each 512-bit block is reduced to 16 words (32 bits) and further replicated to 64 words using the bitwise operations of rotation and shifting. This growth provides the compression function with extra input data in every round, increasing diffusion and ensuring that all bits of the actual message contribute to the final hash value.

Compression and Final Hash Computation: The increased message words in this step are subjected to 64 rounds of bit-wise operations, logical functions and modular additions. In every round, eight working variables are updated through predetermined constants and expanded words. These variables are combined to generate the new hash value when every round is finished. Once every message block has been processed, a final 256-bit hash is produced, which acts as the input message's distinct online fingerprint.

SHA-512 Algorithm

The SHA-512 approach utilizes the advantage of Ron Rivest's asymmetric hash function. This technique builds upon previous algorithms, including SHA-0, SHA-1, SHA-256, and SHA-384. A cryptographic method called SHA-512 creates a 512-bit message digest from any size of communication. The hash algorithm has a collision vulnerability, as shown by its predecessor, SHA1, and MD5, an extension of MD4. The National Institute of Standards and Technology (NIST) has recognized SHA-224, SHA-256, SHA-384, and SHA-512 as the new standard hash functions, as seen in Figure 3. This technique builds upon previous algorithms including SHA-0, SHA-1, SHA-256, and SHA-384. The new standard hash algorithms are SHA-224, SHA-256, SHA-384, and SHA-512, according to the National Institute of Standards and Technology (NIST).

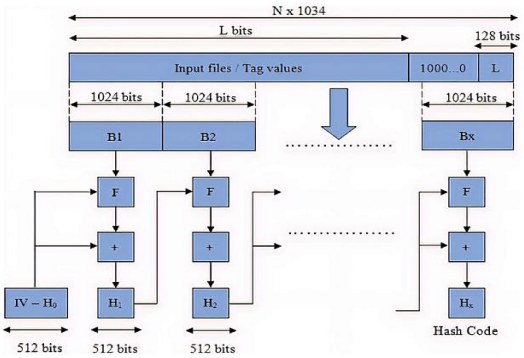

Fig.3 SHA512 Structure

The Addition of Bits

The first step is to add a message with numerous bit wedges to ensure that the message length (in bits) is congruent with 890 mod 1024. The 1024 figure appears because the SHA-512 algorithm looks at messages in blocks of 1024. The message tied to the bundle bits even if it is only 24 bits long. 896- (24 + 1) = 871 bits used to pad the message. Consequently, the wedge bits' lengths vary from 1 to 896. The bits are made up of a bit 1 and the leftover bit 0, which is another item to notice.

Including the Redemption Value for Long Messages

The message is reinserted with 128 bits that match the length of the original message. A message that exceeds 2128 is trimmed modulo 2128. To put it another way, if the message length starts at K bits, the second method adds 128 bits to K modulo 2128, making the message length 1024 bits.

Convert the Hash Value to Initial

In the SHA-512 algorithm, the hash value (0) consists of eight 64-bit hexadecimal words.

Table 1 MD5, SHA-256, and SHA-512 Algorithms: A Comparative Study.

Parameter	MD5	SHA-256	SHA-512
Length of	provides a 128-bit hash value.	Generates a 256-bit hash value,	Generates a 512-bit hash result, which,

Hash (Output Size)	Though each message digest is distinct from its input message, collisions are more likely due to the decreased hash size.	providing significantly better collision resistance and integrity than MD5.	because of its greater bit size, offers extremely good resistance against collision and brute-force assaults.
Message Block Size	Processes data in 512-bit message blocks. Messages shorter than 512 bits are padded to match the required length.	Works on 512-bit message blocks, enabling efficient computation across most modern processors.	Processes input in 1024-bit blocks, making it highly suitable for large-scale, high-security cryptographic applications.
Padding Process	The message is padded to a length of 448 mod 512 before being converted to a 64-bit message. Alignment of blocks is therefore guaranteed.	Padding entails adding a single "1," enough "0s," and then the 64-bit representation of the initial message length.	In order to ensure correct block division, padding adds a 128-bit message length after extending the message to be congruent with 896 mod 1024 bits.
Initialization	Provides the 128-bit message digest by initializing four 32-bit chaining variables (A, B, C, and D) to process data.	Uses eight 32-bit working variables that are updated in 64 iterative rounds to compute the final 256-bit hash.	Initializes eight 64-bit words as hash values (H0–H7), which are updated through 80 rounds of transformations.
Processing / Rounds	Divides each 512-bit block into 16 32-bit words, then processes them through 64 iterations of logical and modular operations.	Executes 64 rounds of bitwise operations, shifts, and additions, providing a balanced trade-off between speed and security.	Performs 80 rounds of similar but more complex compression operations, achieving higher diffusion and security but requiring more computation.
Security & Performance	Very fast but highly vulnerable to collision and preimage attacks; not recommended for modern security applications.	Provides strong security with better performance on 32-bit processors; resistant to most practical attacks.	Offers superior cryptographic strength, optimized for 64-bit processors, and ideal for applications that demand long-term data integrity.
Algorithm Design and Family	Derived from MD4, a message digest algorithm created by Ronald Rivest in 1991. Due to flaws, it is now deemed outdated.	Part of the SHA-2 family developed by NIST; a secure upgrade over SHA-1 with wider adoption in secure protocols like SSL/TLS.	Also, part of the SHA-2 family, designed for 64-bit systems, it improves upon SHA-256 with greater complexity and robustness.
Typical Applications	Commonly used for checksum verification and file integrity checks where high security is not essential.	Widely used in password storage, blockchain, digital signatures, and SSL certificate generation.	Used in high-security contexts such as digital certificates, blockchain technology, and cryptographic authentication systems.

Java Implementation of Hashing Algorithms

Software testing is an Investigation conducted to provide stakeholders with information about the quality of the software product or system under test (SUT). Usually, a software development organization expends between 30% to 40% of total project effort on testing [15] and testing consumes more than 50% of the total cost of a project [16]. A higher-quality software is achieved when SUT is failure-free. A failure is detected When the SUT's external behaviour is different from what is expected of the SUT according to its requirements or some other description of the expected behavior.

The Java implementations utilized the functions of the NetBeans development environment, the Java Cryptography Extension (JCE) and Java Cryptography Application Program Interface (API) for the Java 2 SDK [15]. Through the Java Security Message Digest class, these libraries offer built-in support for secure hash methods, including MD5, SHA-256, and SHA-512. A quick method and automatically get the message digest is to use the generic names for the algorithms (such as "MD5", "SHA-256", or "SHA-512"). The algorithm can be as simple as reading the input data, delivering the digest object via byte arrays or streams, and receiving the hexadecimal hash value back. Because Java is widely compatible, safe from API abuse, and simple to include in schemes where data verification is the primary issue, such as integrity control or password encryption, it is an easy solution for people who desire the system of their choosing [16]. Additionally, NetBeans provides performance measurement that enables programmers to determine the computational cost and execution time of each algorithm in the same environment, providing important information about how the algorithms behave in various system configurations.

Java Cryptography Architecture and Environment Setup

AODV [5] is a single-path on-demand routing protocol for a mobile ad-hoc network. It is composed of two phases; route discovery process and route maintenance process, using nex.

The Java Cryptography Architecture (JCA), which consists of a "provider" architecture and multiple APIs for digital signatures, message digests (hashes), certificates and certificate validation, encryption (symmetric/asymmetric block/stream cyphers), key generation and management, and secure random number generation, is a crucial part of the platform [17]. An object-oriented framework is a set of abstract classes and the connections between their instances that demonstrate a reusable design for any portion of a system. The JCA object-oriented framework provides cryptographic services. The Java Development Kit (JDK) 1.1 initially featured the Java Cryptography Extension (JCE) to offer digital signature and message digest services. Message Authentication Code (MAC) services, encryption, and key agreement were later introduced to JDK 1.2. Unless otherwise noted, the JCA in this article refers to both the original JCA and the JCE extension, as seen in Figure 4.

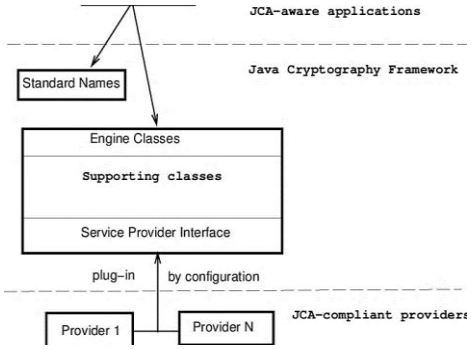

Fig.4 JCA Architecture

Java Keystores (JKS)The first official keystore implementation in Java since JDK 1.2 was released is called Java KeyStore (JKS). When no explicit selection is made, it remains Java 8's default KeyStore. Both public key certificates kept in the clear and encrypted private key entries are supported [18]. The file format, which comes after the list of entries, contains the magic file number, the KeyStore version, and the quantity of entries. A code digest that verifies the KeyStore's integrity makes up the last section of the file. The cryptographic data in each entry is preceded by the object type (key or certificate) and its label.

Java Cryptography Extension KeyStore (JCEKS) The external Java Cryptography Extension (JCE) package now includes Java Cryptography Extension KeyStore (JCEKS) with the release of JDK 1.2. Eventually, starting with version 1.4, it was incorporated into the standard JDK distribution. According to the Java documentation, it "uses much stronger encryption in the form of Password-Based Encryption with Triple-DES," distinguishing it from JKS as a different proprietary KeyStore format. In addition to the enhanced PBE technique, symmetric key storage is made possible.

The first official keystore implementation in Java since JDK 1.2 was released is called Java KeyStore (JKS). When no explicit selection is made, it remains Java 8's default KeyStore. Both public key certificates kept in the clear and encrypted private key entries are supported [19]. The KeyStore version, the number of entries, and the magic file number are all contained in the file format header. Next is the list of things. The file's last portion is a digest that verifies the KeyStore's integrity. The cryptographic data in each entry is preceded by the object type (key or certificate) and its label.

Java Hash-Based Cryptography.

The primary objective of hash-based the purpose of cryptography is to develop digital signature methods that rely on the security of cryptographic hash algorithms, such as SHA-3 [20]. Compared to number-theoretic signature schemes like RSA and DSA, these techniques need fewer security assumptions and rely on the security of hash functions, such as dependence, collision resistance, and resistance to second-preimage attacks [21]. The Merkle Signature Scheme (MSS), which is based on one-time signatures like the Lamport signature scheme, uses a binary hash tree (Merkle tree). The MSS is unaffected by algorithms used in quantum computers. The third phase of NIST standardization produced Sphincs + hash-based signatures as an alternative.

Challenges in Hash Algorithms using Java

The use of hash algorithms in Java also raises several issues regarding performance, security, and compatibility. Among the problems that arise is ensuring that large datasets can be computed efficiently without overloading system resources, because Java's memory management can introduce latency. The reliability of hashing can also be compromised by security weaknesses, such as a weak implementation or poor key management [22]. Furthermore, interoperability between Java versions and between Java and other platforms may be difficult if libraries like Java Cryptography Architecture (JCA) or Java Cryptography Extension (JCE) are used. Debugging and optimization indicate that the code is properly written and verified in terms of speed and cryptographic strength.

• Data-Driven Learning: AI and ML models can learn from vast datasets, extracting patterns and relationships that may be difficult to capture through analytical methods. This enables systems to adapt to complex and non-linear dynamics.

• Real-Time Adaptation: AI algorithms can process data in real time, allowing rapid adjustments to control parameters based on changing conditions. This is crucial for applications where quick responses are essential.

• Robustness: AI-driven adaptive control systems can be more robust to disturbances and uncertainties. They can learn to compensate for unexpected changes, ensuring system stability and performance.

Inadequate access control: This happens when restrictions on what authenticated users can do are not sufficiently enforced.[23], This occurs when restrictions on what authorized users may do are not adequately enforced.

Overflows of Buffers: These components include Web application server components, libraries, drivers, and CGI. Components of web applications designed in languages that fail to properly validate input have the potential to crash and, in certain situations, take over a process [24].

Injection flaws: The external system may carry out malicious commands on behalf of the Web application if an attacker inserts them into the parameters. When web programs visit the local operating system or an external system, they pass parameters.

Denial of service: Attackers may also prevent users from accessing their accounts or cause an application to malfunction. Attackers exploit Web application resources to the extent that other authorized users are unable to access or utilize the program, as was previously described.

Avalanche Criterion Problem: In order to verify the avalanche effect, several message pairs that differ by just one bit must be generated, and their hash values must be determined. Next, each pair of received hashes should have its Hamming distance determined.

Literature Review

This review highlights recent advancements in blockchain and hashing-based security approaches, focusing on techniques such as hash function implementation, FPGA-based optimization, and deep hashing frameworks to enhance data integrity, privacy, and computational performance across various applications.

Jaya et al. (2022) examined and discovered that the hash is one of the blockchain's components. The hash function takes a variable-length input and produces a fixed-length output. A hash approach is used to create transactions with a predetermined size when they are received as input. After that, the hash function is implemented in Java as a blockchain component to protect medical record data during storage and transmission between hospitals using medical record systems [25].

Xu et al. (2022) proposed a two-in-one SM3 algorithm IP core implementation framework for the SM3 hash algorithm. For the compressed iterative part of the SM3 algorithm's core operation, the framework uses a circular, iterative extension approach, adding registers to the combinatorial logic to implement a pipeline. Moreover, the parallelism of hardware is used to improve the addition circuit and reduce the time delay of the longest critical path. The method uses a Virtex-5 FPGA as the target device, and the SM3 algorithm achieves a maximum throughput of 1939.4 Mbps [26].

Umam, Kurniati, and Rahmawati (2021) investigated the impact of interruptions in UML sequence diagram pictures on the Perceptual Hash algorithm's ability to detect visual similarities. Pupils could comprehend the skewed and whirling visual disturbances they observed. These disruptions were tested using digital photos. It demonstrates that visual disruptions have no effect on the Perceptual Hash technique's capacity to assess the similarity of UML sequence diagram pictures. The technique can still detect instances of plagiarism on modified UML sequence diagram pictures with a similarity detection rate greater than 60% [27].

Lagnf and Ganesan (2021) examined the combination of a message counter, SHA256, and AES 128. Unlike other freshness methods that have specific limitations, to preserve freshness, the message counter is encrypted after being uniquely padded to the end of the message. As a consequence, repeated attacks ought to decrease. Thus, creating a hardware design based on an FPGA (Virtex7) is advised. Their real-time hardware design prevents replay attacks, guarantees data integrity and authenticity, and achieves comparatively high throughput and lower payload, according to simulation [28].

Hłobaż (2020) offered a statistical evaluation of the improved SEDEX technique using the SHA-512 hash algorithm. The methodology's statistical analysis is carried out similarly to the SHA-256 approach. In order to do this, the ciphertext files were examined to determine whether they satisfied the pseudo-randomness requirements. A test suite provided by NIST was utilised to evaluate the encrypted data's pseudo-randomness [29].

Zhu, Li and Wang (2019) proposed an additional deep hashing method for situations involving unsupervised picture retrieval. There are two contributions. Initially, the pretrained network's global characteristics are combined to create the pseudo labels, which are then utilised as self-supervised data to maximise the training goal. Second, this deep hashing method uses adaptive feature learning to carry out unsupervised simultaneous hash function and global feature learning [30].

Xenya and Quist-Aphetsi (2019) developed the SHA-256 technique to hash accounting records, protecting sample records that are difficult to change. Furthermore, every compute node would have to update the chain to reflect any effort to modify a record. The technique proved effective at preserving the integrity of forensic account records because of the block's structure, which forbids any data alteration [31].

Table II presents a summary of recent studies on blockchain and hashing algorithms in different domains. The research points out the enhancement of data integrity, security, and performance in various ways, such as FPGA-based designs and deep hashing. Nonetheless, scalability and validation in actual conditions are still major problems for most studies. The direction for future work is clear, that is, to improve efficiency, broaden usage, and introduce hybrid cryptographic techniques.

Table 2 Summary of Recent Studies on Blockchain and Hashing-Based Security Approaches

Reference	Study On	Approaches	Key Findings	Challenges / Limitations	Future Directions
Jaya et al. (2022)	Secure storage and sharing of medical records using blockchain	Implementation of hash function in Java to generate fixed-length outputs for securing transactions	Successfully ensured data integrity and privacy during storage and exchange among hospitals	Implementation limited to Java and tested in a controlled environment only	Expand implementation to multi-platform systems and integrate with real-time medical data exchange
Xu et al. (2022)	High-throughput	Two-in-one SM3 algorithm IP core with	Achieved a maximum throughput of 1939.4	Limited to Virtex-5 FPGA platform;	Extend to newer FPGA architectures and enhance

	SM3 hash algorithm hardware design	circular iterative pipeline and FPGA-based optimization	Mbps, reducing critical path delay	scalability to newer devices not evaluated	energy efficiency
Umam, et.al. (2021)	Detecting similarity in UML sequence diagrams	Perceptual Hash algorithm tested on rotated and skewed images	The algorithm maintained similarity detection >60% accuracy even with distortions	Limited to specific types of image distortions and synthetic test cases	Apply method to real-world student submissions and integrate more complex distortions
Lagnf et.al. (2021)	Prevention of replay attacks and ensuring data authenticity	Integration of AES-128, SHA-256, and message counter using FPGA (Virtex-7)	Ensured data integrity, authenticity, and high throughput with reduced payload	Focused mainly on hardware-level validation without large-scale network testing	Implement real-time system testing under varied network environments
Hłobaż et.al.(2020)	Statistical validation of enhanced SDEx encryption	SHA-512-based hash function with NIST pseudo-randomness testing	Ciphertext passed statistical tests ensuring strong pseudo-randomness	Comparison limited to SHA-256; lack of broader algorithmic benchmarking	Extend analysis to hybrid cryptographic methods combining multiple hash layers
Zhu, Li et.al. (2019)	Unsupervised image retrieval using deep learning	Deep hashing framework with pseudo-label generation and adaptive feature learning	Enabled self-supervised hash function learning and efficient feature representation	Computationally expensive and limited to image domain	Extend to multimodal retrieval (image-text, video) and improve scalability
Xenya et.al. (2019)	Blockchain-based forensic accounting data security	Application of SHA-256 hashing for immutable record keeping	Efficiently maintained data integrity and prevented unauthorized alterations	Focused mainly on theoretical blockchain framework	Implement on real forensic systems and evaluate transaction scalability

Conclusion and Future Work

Hashing methods that are popularly found in the form of MD5, SHA-256, and SHA-512, have been considered at an utmost level and have been working as the backbone of modern computer systems for powerful security in the aspects of data integrity and authentication. The thorough comparison of the three above-mentioned algorithms has shown that although all of them are very much necessary for the purpose of securing the data and the users verifying their identity, their efficiency and security vary significantly. The Java-based implementation served as a constant and controlled environment for the study where the algorithms, in turn, went through the trials of efficiency, computational costs, and cryptographic resilience. MD5 is a powerful hash for quickly hashing files, but it can easily be undone, and therefore it is not fit for modern cryptographic applications. SHA-256 is a little bit slower than MD5 but the difference is not so much and that speed along with the strength it has makes it perfect for use in blockchain and authentication systems. SHA-512, though it brings a higher computational burden, it is also more resistant to cryptographic attacks like brute-force and collisions and that is why it is widely used in high-security places where those features are critical. The findings indicate that the algorithm should be chosen with the security and resource aspects in mind. Besides, the revelations of the study are strong enough to convince the transition from the outdated hashing functions to the widely accepted ones like SHA-256 and SHA-512 so as to win the battle of long-term data loss protection.

Future work must go beyond this comparative study to assess the security posture of the new hash functions, e.g., SHA-3 and BLAKE2, and also that of the existing ones. To make the results more convincing, hardware-based implementations using FPGA or GPU acceleration could be used to determine energy efficiency and real-time performance when processing large sets of data. It should be mentioned that prediction models based on machine learning may also be utilised to dynamically select the optimal hash algorithms for specific application scenarios. Besides, using hybrid cryptographic frameworks that consist of multiple hash layers could be a more effective way to ensure cybersecurity as these threats continue to change.

References

[1] F. Martín-Fernández and P. Caballero-Gil, "Analysis of the new standard hash function," 2022. doi: 10.48550/arXiv.2209.11857.

[2] M. C. A. Kioon, Z. Wang, and S. D. Das, "Security Analysis of MD5 Algorithm in Password Storage," Appl. Mech. Mater., vol. 347–350, 2013, doi: 10.2991/isccca.2013.177.

[3] M. Sumagita and I. Riadi, "Analysis of Secure Hash Algorithm (SHA) 512 for Encryption Process on Web-Based Application," vol. 7, pp. 373–381, 2018.

[4] V. Schmitt and J. Jordaan, "Establishing the Validity of MD5 and SHA-1 Hashing in Digital Forensic Practice in Light of Recent Research Demonstrating Cryptographic Weaknesses in these Algorithms," vol. 68, no. 23, pp. 40–43, 2013.

[5] H. P. Kapadia, "AI Enhanced Web Accessibility Features," vol. 8, no. 4, pp. 476–483, 2021.

[6] S. Debnath, A. Chattopadhyay, and S. Dutta, "Brief review on journey of secured hash algorithms," in 2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix), 2017, pp. 1–5. doi: 10.1109/OPTRONIX.2017.8349971.

[7] A. Zniti and N. El Ouazzani, "A comparative study of hash algorithms with the prospect of developing a CAN bus

authentication technique," Int. J. Electr. Comput. Eng. Syst., vol. 13, no. 9, pp. 741–746, Dec. 2022, doi: 10.32985/ijeces.13.9.2.

[8] Y. Zhang, Y. Xiao, M. M. A. Kabir, D. D. Yao, and N. Meng, "Example-based vulnerability detection and repair in Java code," in Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, May 2022, pp. 190–201. doi: 10.1145/3524610.3527895.

[9] R. Sobti and G. Geetha, "Cryptographic Hash Functions: A Review," Int. J. Comput. Sci. Issues, vol. 9, no. 2, 2012.

[10] E. V. Maliberan, A. M. Sison, and R. P. Medina, "A New Approach in Expanding the Hash Size of MD5," Int. J. Commun. Networks Inf. Secur., vol. 10, no. 2, pp. 374–379, Apr. 2018, doi: 10.17762/ijcnis.v10i2.3292.

[11] S. Das, S. De, and R. Kumar, "Implementation and Comparative Analysis of RSA and MD5 Algorithm," Int. J. Comput. Appl., vol. 141, no. 9, pp. 10–13, May 2016, doi: 10.5120/ijca2016909789.

[12] B. Meena and P. M, "Comparative Analysis of Cryptographic Hash Algorithms," Int. J. Res. Appl. Sci. Eng. Technol., vol. 10, no. 5, pp. 4362–4368, May 2022, doi: 10.22214/ijraset . 2022.43370.

[13] S. M. Myint, M. M. Myint, and A. A. Cho, "A Study of SHA Algorithm in Cryptography," vol. 3, no. 5, pp. 1453–1454, 2019.

[14] R. Wu, X. Zhang, M. Wang, and L. Wang, "A High-Performance Parallel Hardware Architecture of SHA-256 Hash in ASIC," in 2020 22nd International Conference on Advanced Communication Technology (ICACT), IEEE, Feb. 2020, pp. 1242–1247. doi: 10.23919/ICACT48636.2020.9061457.

[15] G. A. Francia and R. R. Francia, "An Empirical Study on the Performance of Java/.Net Cryptographic APIs," Inf. Syst. Secur., vol. 16, no. 6, pp. 344–354, Dec. 2007, doi: 10.1080/10658980701784602.

[16] N. A. Fauziah, E. H. Rachmawanto, and D. R. I. M. Setiadi, "Design and Implementation of AES and SHA-256 Cryptography for Securing Multimedia File over Android Chat Application," in International Seminar on Research of Information Technology and Intelligent Systems (ISRITI) Design, 2018. doi: 10.1109/ISRITI.2018.8864485.

[17] S. Rahaman et al., "CryptoGuard," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, ACM, Nov. 2019, pp. 2455–2472. doi: 10.1145/3319535.3345659.

[18] R. Focardi, F. Palmarini, M. Squarcina, G. Steel, and M. Tempesta, "Mind Your Keys? A Security Evaluation of Java Keystores," in Proceedings 2018 Network and Distributed System Security Symposium, 2018. doi: 10.14722/ndss.2018.23083.

[19] A. De Caro and V. Iovino, "jPBC: Java pairing-based cryptography," in 2011 IEEE Symposium on Computers and Communications (ISCC), 2011, pp. 850–855. doi: 10.1109/ISCC.2011.5983948.

[20] S. Rahaman et al., "CryptoGuard: High Precision Detection of Cryptographic Vulnerabilities in Massive-sized Java Projects," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, Nov. 2019, pp. 2455–2472. doi: 10.1145/3319535.3345659.

[21] C. Balamurugan, K. Singh, G. Ganesan, and M. Rajarajan, "Post-Quantum and Code-Based Cryptography—Some Prospective Research Directions," Cryptography, vol. 5, no. 4, 2021, doi: 10.3390/cryptography5040038.

[22] V. Akoto-adjepong, M. Asante, and S. Okyere-gyamfi, "An Enhanced Non-Cryptographic Hash Function," Int. J. Comput. Appl., vol. 176, no. 15, pp. 10–17, 2020.

[23] R. R. Dewanagn, D. Thombre, and K. Sharma, "Implementation of Secure Hash Algorithm Using JAVA Programming," Int. Res. J. Eng. Technol., vol. 2, pp. 528–533, 2015.

[24] B. R. Cherukuri, "Microservices and containerization: Accelerating web development cycles," World J. Adv. Res. Rev., vol. 6, no. 1, pp. 283–296, Apr. 2020, doi: 10.30574/wjarr . 2020.6.1.0087.

[25] T. Jaya, S. E. Nugraha, G. R. Davinsi, J. V. Moniaga, and B. A. Jabar, "Implementation of Blockchain Technology Using Hash Method in Java for Medical Record Application Development," in 2022 IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA), 2022, pp. 1–6. doi: 10.1109/ICITDA55840.2022.9971286.

[26] Y. Xu, L. Han, Z. Yu, and F. Che, "Optimized Design Implementation and Research of SM3 Hash Algorithm Based on FPGA," in 2022 2nd International Conference on Computer Science and Blockchain (CCSB), 2022, pp. 111–117. doi: 10.1109/CCSB58128.2022.00027.

[27] K. Umam, S. Kurniati, and P. N. A. Rahmawati, "The Effect of Disturbance in UML Sequence Diagram Images on Image Plagiarism Detection Ability Using the Perceptual Hash Algorithm," in 2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS), 2021, pp. 1–5. doi: 10.1109/ICORIS52787.2021.9649542.

[28] F. M. E. Lagnf and S. Ganesan, "Securing CAN FD by implementing AES-128, SHA256, and Message Counter based on FPGA," in 2021 IEEE International Conference on Electro Information Technology (EIT), 2021, pp. 91–96. doi: 10.1109/EIT51626.2021.9491920.

[29] A. Hłobaż, "Statistical Analysis of Enhanced SDEx Encryption Method Based on SHA-512 Hash Function," in 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1–6. doi: 10.1109/ICCCN49398.2020.9209663.

[30] Y. Zhu, Y. Li, and S. Wang, "Unsupervised Deep Hashing With Adaptive Feature Learning for Image Retrieval," IEEE Signal Process. Lett., vol. 26, no. 3, pp. 395–399, 2019, doi: 10.1109/LSP.2019.2892233.

[31] M. C. Xenya and K. Quist-Aphetsi, "A Cryptographic Technique for Authentication and Validation of Forensic Account Audit Using SHA256," in 2019 International Conference on Cyber Security and Internet of Things (ICSIoT), 2019, pp. 11–14. doi: 10.1109/ICSIoT47925.2019.00008.