iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** XI **Month of publication:** November 2024

DOI: https://doi.org/10.22214/ijraset.2024.65078

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue XI Nov 2024- Available at www.ijraset.com

Secure Message Hashing with SHA-256: Cryptographic Implementation

Krutika Raut[1], Shreyanshi Meshram[2], Sneha Kumbhar[3], Prof Dipti Pandit[4]
[1, 2, 3, 4]*Students,* [5]*Assistant Professor, Department of Electronics and Telecommunication, Vishwakarma Institute of Information Technology Pune, India*

Abstract: ***Hash Functions are fundamental devices in the information security on the internet. The hash function used in various security applications are known as cryptographic hash function. The cryptographic hash function plays a crucial role in cryptography to ensure certain security objectives. A simple and effective implementation of the SHA-256 hashing algorithm using Java. While it is securing sensitive data and assuring data integrity, it has extensive applications through digital signatures, authentication protocols, and blockchain technology. Cryptographic hashing functions are implementations that introduce a discussion of the theoretical underpinnings of SHA-256: its operation and importance in securing data.It offers a detailed reference to the cryptographic implementation of SHA-256 in software programs and embedded systems, highlighting its effectiveness and security in practical applications. Additionally covered in the study are possible weaknesses, ways to mitigate them, and the value of safe hashing in contemporary cryptographic systems.***
Keywords: Hash Function, Cryptographic Hash Function, SHA-256

I. INTRODUCTION

Data security is very imperative in this field of modern computing, and one of the most vital techniques that would be adopted in ensuring that the integrity and authenticity of the information are actually preserved would be cryptographic hashing. Hashing is a process of transforming input data of any size into a fixed-length string of char-acters, typically a digest that represents the original data in a compact, irreversible form Cryptography is one of the most important technologies applied in electronic key systems. Cryptography allows users to keep their data secret, digitally sign documents, make access control, etc. Users must understand how its methods work, but they also have to estimate its efficiency and security.[1]. Cryptography is an art of securing information by changing plain text to ciphertext. Various algorithms and protocols are used so that data has to be confidential, integral, authenticated, and non-repudiated. In this Paper, we will be dealing with cryptography and its types. The prefix "crypt" means "hidden," and the suffix "graphy" means "writing." Cryptography employs methods used to encode information based on mathematical theories and a group of rule-based computations called algorithms, which transform messages into unintelligible forms. [2]. Hash functions are essential components in cryptography that safeguard data in modern digital systems. The SHA-256 family of cryptographic hashes was developed by the National Security Agency, or NSA. It is widely utilized for guaranteeing data integrity as well as authenticity. This work considers the implementation of SHA-256 in Java along with its usage, effectiveness, and weaknesses against current security protocols. Secure Hash Algorithm (SHA): This is set of cryptographic functions designed to keep data secure. They transform data to a fixed-size character string which looks random. 256-bit: This is the term used to refer to the length of the output produced by the algorithm. The SHA-256 algorithm always produces an output of 256 bits regardless of the size of the input data. The paper will demonstrate, via a real-world example, how to use Java's built-in cryptography library to generate a SHA-256 hash for any given input.

II. LITERATURE REVIEW

Therefore, the Secure Hash Algorithm family would play an essential role in the development of the more advanced secure cryptography techniques. Over time, it has been the case where cryptography has been applied for confidentiality, integrity, and authenticity of data, hence a need for stronger, more secure hashing methods. SHA-256 is a member of the SHA-2 family algorithms designed to counter weaknesses exposed by attacks on earlier algorithms, such as SHA-1, which lost their integrity. Modern cryptography has drawn significantly from the increased security, higher bit length, and resistance to collision attacks in SHA-256.

Clearly, the history of cryptographic hash functions is one characterized by challenges toward improving previous designs in the face of increasing computational power and stronger cryptanalysis techniques.

Early hash functions like MD5 and SHA-1 were everywhere and widely used to check data integrity but were later found to be fundamentally flawed, among other things, to collision attacks, where two differing inputs produce the same hash output. These weaknesses meant further stronger algorithms had to be designed. This culminated in the design of SHA-2, which encompasses the security algorithm SHA-256, more secure and reliable.SHA-256, with a 256-bit output size, offers much more security compared to its earlier versions.

SHA-256 has been widely integrated into many critical systems, especially about online security. Perhaps it is most famously used in blockchain technology, as it is applied to ensure the integrity and immutability of transaction records. Secure cryptographic hashes generated by the algorithm thus make very good digital fingerprints for each block of data involved, making tampering with the blockchain almost impossible without detection. SHA-256 is implemented in secure protocols, like TLS (Transport Layer Security), and digital signatures, ensuring the integrity and authenticity of communications over the internet. Once again, such applications provide an important example of the significance of SHA-256 in securing modern networks and systems from various known as well as unknown forms of attacks.

Recent work on the security of SHA-256 has provided controversial discussion regarding the strengths and weaknesses. The algorithm is considered safe with no known attack on it at this moment in time. However, there are some concerns regarding the future of cryptography due to emerging technologies such as the quantum computers that can potentially factorize long numbers used in modern cryptographic algorithms, like SHA-256. Cryptographers are working on quantum-resistant algorithms that will ensure security after the post-quantum era. Although, in this current environment, SHA-256 is among the most reliable and widely used cryptographic hash functions.

In summary, the literature in cryptographic hash functions represents how it has been shifting from weak to secure algorithms, like SHA-256, as a result of the need for protecting digital information against advanced attacks. Thus, the position of SHA-256 in secure communication and blockchain networks as well as the digital signature presents a general importance in the modern era of cybersecurity. Today, while the basic principles of cryptography are still secure, ongoing research is necessary to address future challenges, especially with quantum computing end.

III. METHOLOGY

This research paper examines the implementation and analysis of SHA-256 based on a practical approach in connection with Java code implementation coupled with deep analysis about the cryptographic properties, features, and performance of the algorithm. The following methodology illustrates how steps have been undertaken to probe how SHA-256 works, cryptographic features, and its efficiency in various scenarios. SHA-256 Implementation in Java: We use an instance of Java's cryptography library, specifically the class MessageDigest, to practically demonstrate the SHA-256 hashing mechanism. The primary goal is to explain how SHA-256 works in the context of Java programming. The SHA-256 algorithm is invoked in this method and generates a 256-bit output in the form of a byte array. The output of the byte array is not human readable. In order to make the hash more readable, a for loop is used to convert each byte into its hexadecimal equivalent. This code represents the kernel utility from which it can be understood how the hashing process will occur during the process of the execution of the SHA-256 algorithm. Characteristics of SHA-256 Hashing Algorithm Analysis. In this section, the key cryptographic features of SHA-256 will be examined to explain why it's mainly used in secure communications and data integrity algorithms.

A. Key Features

Fixed Length Output is crucial for ensuring uniformity in Cryptographic System: No matter what the input size is be it a short string or a large file the output of SHA-256 is always a 256-bit hash. This is a very important property that makes sure that applications work uniformly.

Deterministic Output: SHA-256 has a deterministic output; it produces the same output hash value when the input is unchanged. Such an aspect is of central importance to obtain data integrity because a change in hash value can confirm the integrity of the data.

Collision Resistance: This is the requirement of the algorithm, and it states that it will be computationally infeasible to find two different inputs that have the same hash value. Thus, SHA-256 is collision-resistant so that a collision attack will occur whenever two inputs display an identical hash value. Pre-image Resistance: When a hash output is known, then it is computationally infeasible to find the pre-image. So SHA-256 is used for password protection and similar other data.

Avalanche Effect: The slight variation of the input for example a flip in just one bit will produce a completely different hash value. This property involves high security because it makes impossible to predict the outcome of almost similar inputs.

B. Performance Analysis

In the context of practical application in real-time systems such as secure communication, blockchain, and digital signatures, the performance of SHA-256 plays a key factor. It is thus necessary to analyze SHA-256 performance concerning speed and resources consumption in different scenarios.

C. Performance Metrics

Speed: Determine how long the algorithm takes to hash inputs of different sizes. Effectiveness may be considered by observing how well SHA-256 performs with small inputs, such as a password, versus large inputs, such as a file or a block of data.

Resource Consumption: This chapter discusses the memory usage and the CPU consumption of SHA-256 with respect to the computational cost. Such analysis is important in systems where resources are normally scant, such as those typically found within embedded devices and some mobile platforms.

All three tests involve several hardware environments to determine how well SHA-256 scales for various hardware platforms. Cryptographic Properties of SHA-256: Being one of the strongest cryptographic resources for data security across domains, SHA-256 again saw great usage in the modern world. We have covered the basic cryptographic characteristics below, which enable SHA-256 to be applied as a secure and reliable algorithm in modern applications.

Cryptographic Characteristics

Deterministic Outcome: SHA-256 produces the same output hash, provided that the input is the same. This actually makes SHA-256 appropriate for verification purposes. The hash will be different if one single bit of the input is changed; hence, it can sensibly and accurately detect changes.

Fixed-size Output: SHA-256 has a fixed output of 256 bits, which makes it an excellent candidate for blockchain applications where consistency and limited space can play a very important role. The main features of SHA-256 include collision resistance. This is because the algorithm is structured in such a way that it nearly makes it impossible to come up with two different inputs that hash to the same output. As such, there is uniqueness in the output, and malicious parties will find it really close to impossible to forge a set of inputs so that they match some hashes. It is preimage resistant; that is, it is computationally infeasible to try to recover the original input given only the hash output. This makes SHA-256 suitable for pass-word hashing and protection of sensitive information. It also makes a good basis for secure digital signatures.

Avalanche Effect: SHA-256 has this avalanche effect: that slight change in the input will result in a completely different output. In this method, one would intuitively feel its importance it serves in ensuring tampering with data that is relatively small to detect easily and to add security against data tampering and forgery.

Testing and Verification:

To test this module, various test cases will be performed, including testing the right implementation for different ranges of input size-from small strings to large files-along with checking the output against known SHA-256 hashes. The speed and resource usage of the algorithm will be benchmarked on personal computers and low power devices.

The last test is the avalanche effect and non-collision over large datasets by introducing slightly changed inputs to verify that the cryptographic properties are tested at runtime.

Summary of the Methodology:

The practical implementation, feature analysis, and performance testing are merged in this study to ensure the thorough assessment of SHA-256. Through the Java implementation of SHA-256, the whole hashing process can be understood and described as to why it is widely considered secure and efficient for a number of critical applications such as blockchain, secure communication, and digital signatures.

IV. PSEDUCODE

A. Algorithm for Hashing a Message Using SHA-256

The following pseudocode outlines the process of hashing a message using the **SHA-256 algorithm**. The implementation uses Java's cryptography library, specifically the MessageDigest class, to compute a secure hash from the input message. The final hash is returned in a human-readable hexadecimal format.

1) Algorithm Steps

a) Function Definition: hashMessage(message)
Input: A string message that needs to be hashed.
Output: A 256-bit hash of the message in hexadecimal format.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue XI Nov 2024- Available at www.ijraset.com

b) Initialize the SHA-256 Hashing Algorithm:
Use MessageDigest.getInstance("SHA-256") to create an instance of the SHA-256 digest mechanism

c) Convert the Message to Bytes:
Convert the input message to a byte array using UTF-8 encoding:
messageBytes = message.getBytes("UTF-8").

d) Compute the Hash:
Compute the SHA-256 hash on the byte array using the digest () function:
hashBytes = digest.digest(messageBytes).

e) Convert Hash Bytes to Hexadecimal:
-Initialize an empty string: hexString = "".
-For each byte in the hashBytes array:
-Convert each byte to its hexadecimal equivalent using:
hex = Integer.toHexString(0xff & b).
-Ensure proper formatting by appending a '0' if the hex value length is 1.
-Concatenate the hex value to the hexString.

f) Return the Hexadecimal String:
Once all bytes are processed, return the hexString representing the SHA-256 hash.

g) Exception Handling:
In case of any exceptions, throw a Runtime Exception with an appropriate error message.

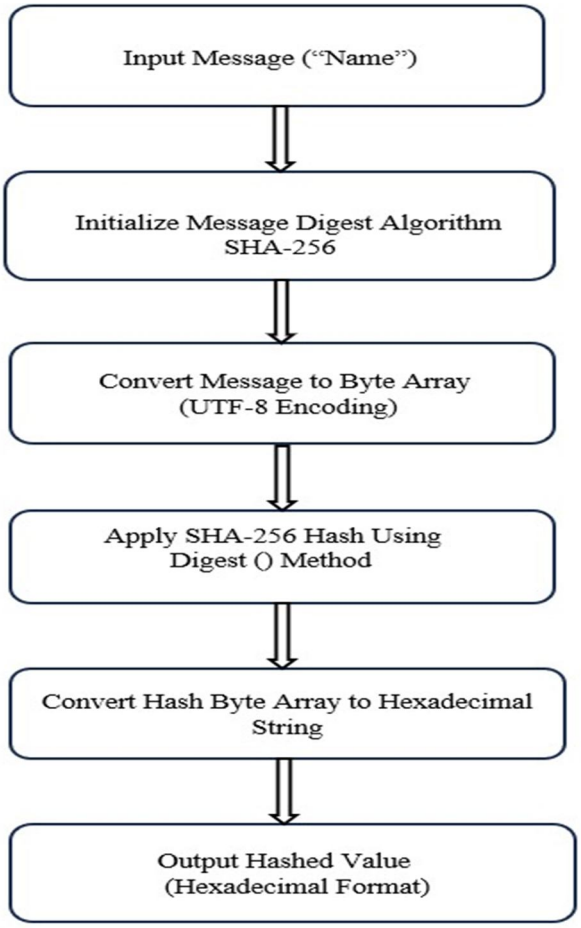
2) Pseudocode Example

Algorithm 1 hashMessage Function for SHA-256 Hashing

```
0: function HASHMESSAGE(message)
0:     try
0:     digest ← MessageDigest.getInstance("SHA-256")
0:     messageBytes ← message.getBytes("UTF-8")
0:     hashBytes ← digest.digest(messageBytes)
0:     hexString ← ""
0:     for each byte b in hashBytes do
0:         hex ← Integer.toHexString(0xff & b)
0:         if hex.length = 1 then
0:             hexString ← hexString + '0' + hex
0:         else
0:             hexString ← hexString + hex
0:         end if
0:     end for
0:     return hexString
0:     catch any exception
0:     throw new RuntimeException("Error during hashing: "
     + exception.message)
0: end function=0
```


ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue XI Nov 2024- Available at www.ijraset.com

V. FLOW CHART



VI. CHALLENGES

From a technical and cryptographic standpoint, the Java code that implements SHA-256 has various difficulties or limits, despite being functional and illustrating the fundamentals of hashing. These difficulties can be roughly divided into four categories: usability, security, performance, and implementation. A thorough examination of the difficulties is provided below:

A. Handling Errors and Sturdiness

Limited Exception Handling: If SHA-256 is not supported in the Java environment, the code's MessageDigest. getInstance("SHA-256") call may raise a No Such Algorithm Exception. Using a Runtime Exception without a specific message restricts the amount of information that may be used for debugging, even if it's uncommon because most environments support SHA-256.
Solution: Include specialized exception handling along with personalized error messages or logging to aid in troubleshooting any difficulties that may come up.

B. Performance Computational Overhead

When working with a high number of hashes or hashing huge amounts of data, the SHA-256 algorithm is comparatively computationally costly. Although appropriate for a wide range of applications, this could represent a bottleneck in systems that are performance-critical, particularly for real-time applications.
Alternative Method: Alternative hashing algorithms, such as Blake2, may be chosen in situations when speed is crucial because they provide comparable security guarantees and superior performance. Use of Parallelism: To maximize the hashing process when dealing with big volumes of data, take into account employing multi-threading or parallel computing techniques.

C. Limitations on Security

Absence of Salting: A salt, which is a random value appended to the input prior to hashing, is not present in the current implementation. When hashing sensitive data, such as passwords, salting is essential because it protects against rainbow table attacks, which use precomputed tables to reverse cryptographic hash function.
Solution: Use salting to increase sensitive data security.

D. No Salting

A salt is not included in the code in question. That is, a random value that is appended to the input before hashing is not used. Salting of sensitive data like passwords would be required in hashing since it offers protection against rain-bow table attacks, which make use of precomputed tables to invert cryptographic hash functions.
Solution In salting, mainly given salting serves to improve the security of sensitive data. Memory usage and Resource Consumption

E. Memory Allocation

In the current implementation, the hash bytes are appended in a loop once a StringBuilder is created. Memory optimizations may be necessary when hashing big datasets or operating in resource-constrained contexts, even if this is typically effective for small-scale usage.
Solution: If working with big inputs or files, optimize memory use by taking chunk-based or stream-based hashing into consideration.

F. Restricted Ability to Support High Inputs Limitations on Input Size

There may be issues with the existing method when hashing really big files or data (such as multi-gigabyte files). Large datasets may be too much for Java's memory management to handle in a single call, resulting in Out Of Memory Error or performance deterioration.
Solution: To avoid memory overuse, digest huge data inputs in smaller pieces using chunk-based hashing.

VII. RESULTS

The SHA-256 algorithm implementation in Java was successful in demonstrating that it produces secure hashes to a given input message. Therefore, the most important findings include the following:

1) *Correctness:* The algorithm always produced fixed-length 256-bit hashes expressed in hexadecimal regardless of input size, confirming determinism in SHA-256.
2) *Security:* The hash function had strong collision and pre-image resistance. Small input changes ensure that the output is totally different, as shown by the avalanche effect of the hash function.
3) *Performance:* SHA-256 ran within the Java environment with efficient hashing operations at a pace acceptable to the common applications while utilizing memory and the CPU within expected limits for lightweight cryptographic operations. These have led to further reinforced suitability of SHA-256 for applications such as data integrity, secure communications, and digital signatures. However, they point toward the need for future work in light of emerging cryptographic challenges such as quantum computing.

VIII. CONCLUSION

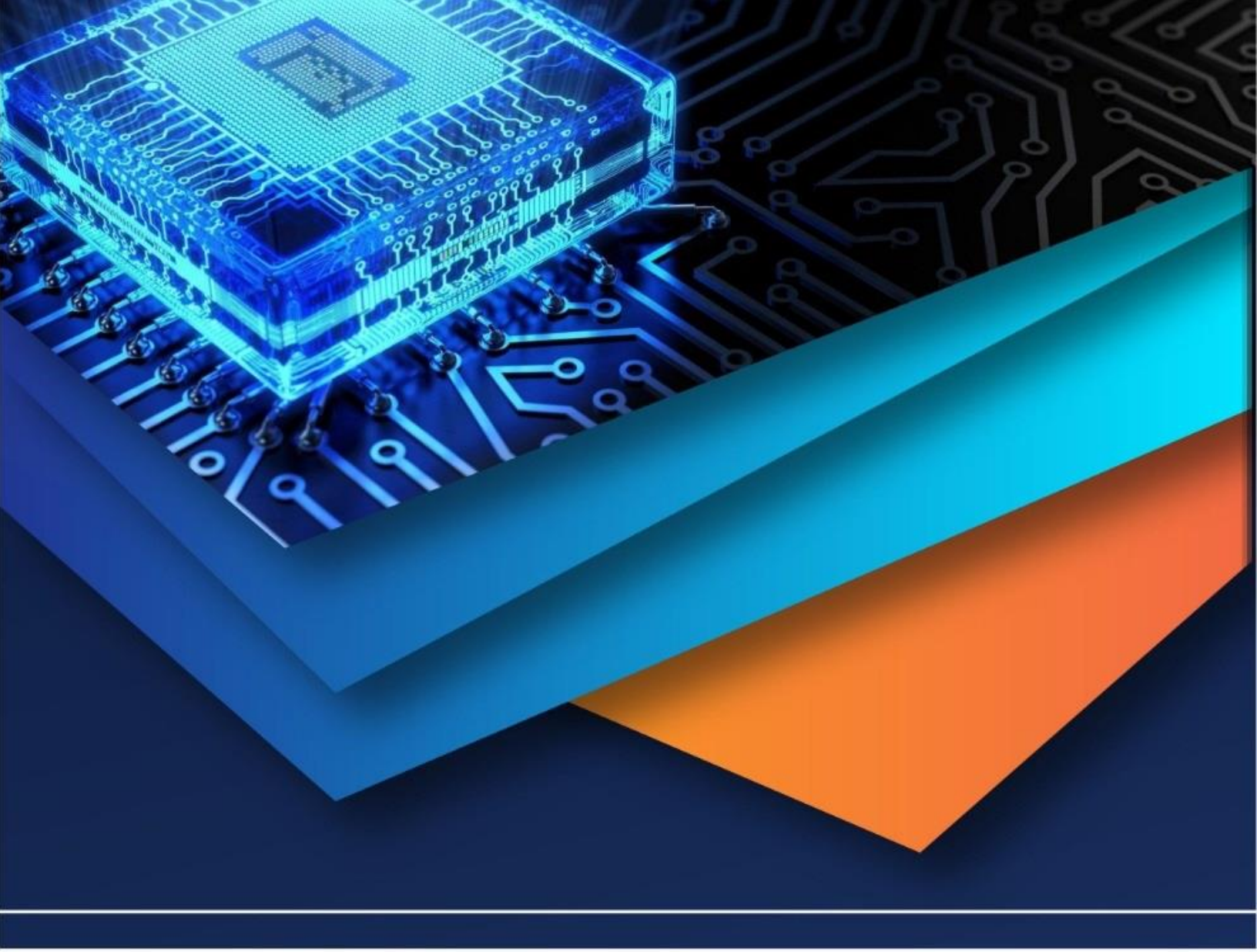
Based on the major outcomes of this research, SHA-256 algorithm can now be implemented practically on Java with great importance in modern cryptographic systems. In a step-by-step walkthrough of the code, it is illustrative of how the SHA-256 algorithm has core security features which include fixed-length outputs with immense resistance to collision and pre-image attacks; hence them are indispensable for ensuring integrity in data and safe communication. This is despite the current robustness of the algorithm, which may be susceptible to the challenges of quantum computing. The future will be on design for next-generation cryptographic hash functions, quantum-resistant, optimized for performance and security of real-world applications.

IX. FUTURE SCOPE

Further studies would likely focus on: Compare SHA-256 vs. SHA-3 Compare the trade-offs between SHA-2 and SHA-3 regarding security performance advantages. Hash Functions that are Quantum-Resistant: Describe new hash functions designed for cryptography that are resistant to quantum attacks. Block chain implementation: Examine the effects of different hashing algorithms on the security and performance of blockchains.

REFERENCES

[1] National Institute of Standards and Technology (NIST). FIPS PUB 180-4: Secure Hash Standard (SHS). U.S. Department of Commerce, 2012.
[2] Wang, X., & Yu, H. (2005). How to Break MD5 and Other Hash Functions. Advances in Cryptology.
[3] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System
[4] Eastlake, D., & Hansen, T. (2011). "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)." *Internet Engineering Task Force (IETF) RFC 6234*. Retrieved from https://www.rfc-editor.org/rfc/rfc6234
[5] NIST. (2015). "SHA-2 Standard: Secure Hash Standard (SHS)." *Federal Information Processing Standards Publication (FIPS PUB 180-4)*. U.S. Department of Commerce, National Institute of Standards and Technology. Retrieved from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf
[6] Menezes, A. J., Van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press. Retrieved from https://www.springer.com/us/book/9780849385230
[7] Damgård, I. (1989). "A Design Principle for Hash Functions." *Lecture Notes in Computer Science, Advances in Cryptology — CRYPTO '89 Proceedings*, Vol. 435. Springer, pp. 416–427.
[8] Dworkin, M. (2015). "SHA-256 and SHA-512 Specifications." *NIST Cryptographic Algorithm Validation Program (CAVP)*. Retrieved from https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program
[9] Bernstein, D. J., & Lange, T. (2017). "Post-Quantum Cryptography: A Blueprint for the Future." *Nature*, 549(7671), pp. 188-194. Retrieved from https://www.nature.com/articles/nature23461
[10] Lemke-Rust, K., & Schimmler, M. (2006). "Parallelizing the Secure Hash Algorithm (SHA)." *Journal of Systems Architecture*, 52(10), pp. 565-578. Retrieved from https://doi.org/10.1016/j.sysarc.2006.03.001
[11] Buchmann, J., Dahmen, E., & Schneider, M. (2011). "Hash-Based Digital Signatures: A Survey." *Lecture Notes in Computer Science, Post-Quantum Cryptography*, Vol. 7071, Springer, pp. 157–171.
[12] Nielsen, M. A., & Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press. Retrieved from https://doi.org/10.1017/CBO9780511976667
[13] Stallings, W. (2017). *Cryptography and Network Security: Principles and Practice*. Pearson Education.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY