SHARAN-256: A Modified SHA-256 Hashing Technique for Enhanced Security

Anant Jain[1][0009−0000−2859−8911] ritujainanant2810@gmail.com, Gauranshi Gupta[1][0009−0001−9918−3120]gauranshigupta2000@gmail.com , and Rahul Johari*[1][0000−0002−7675−8550]rahul@ipu.ac.in,

Deo Prakash Vidyarthi[3][0000−0003−4151−0552]dpv@mail.jnu.ac.in

[1] SWINGER : **S**ecurity, **W**ireless, **I**oT **N**etwork **G**roup of **E**ngineering and **R**esearch, University School of Automation and Robotics(USAR), Guru Gobind Singh Indraprastha University, Delhi, India
[2] School of Computer and Systems Sciences, Parallel and Distributed System Lab JNU, Delhi, India

Abstract. This paper introduces SHARAN-256, a novel hashing algorithm derived from SHA-256, which focuses on enhancing security while maintaining a balance between space and time complexity. SHARAN-256 strengthens resilience against malicious attacks. An important aspect of SHARAN-256 is the integration of Random Number Generation (RNG) techniques by creating random numbers through various mathematical operations with input messages. It also utilizes parallel processing techniques, using thread pools to reduce computational time. The study compares SHARAN-256 with SHA-256, highlighting its capability to provide more security. By introducing a modified SHA-256 algorithm enhanced by RNG and parallel processing capabilities, this research significantly advances cryptographic methodologies, safeguarding sensitive data in contemporary digital environments.

Keywords: Security · Hashing · SHA-256 · Cryptography

1 Introduction

Hashing is a foundational concept in computer science, transforming any type of data, be it text or files, into a fixed-length sequence of characters including random strings of letters and numbers. This resulting string is referred to as a "hash" just like a fingerprint for the data, providing an identifier. The key feature of hashing is its ability to generate a unique hash value for each distinct set of input data, producing a drastic change in the hash even with minor alterations in the input. A hash function is responsible for carrying out the hashing process. This function takes input data and applies a series of mathematical operations to produce the fixed-length hash. A hash function is a one-way function, which

* corresponding author

means that it is computationally infeasible to reverse the process and derive the original input from the hash. One primary use of hashing is in ensuring data integrity, where the hash serves as a checksum or digital fingerprint.

1.1 Applications of Hashing

Hashing techniques are used to securely store passwords by converting them into hash values, protecting sensitive information from unauthorized access. By generating a unique hash value for a dataset, any changes in the data can be quickly detected by comparing the original hash with the recalculated hash. Each block in a BlockChain contains a hash of the previous block, creating an unbreakable chain of blocks. Additionally, cryptographic hash functions are used to secure transactions, generate addresses, and create unique identifiers for blocks.

1.2 Description about SHA256

1. Message Padding: Padding is done to make the message length a multiple of 512. A '1' bit is appended, followed by a series of '0' bits, and finally the original message length in bits.

2. Initialization: Eight 32-bit variables (A, B, C, D, E, F, G, H) with specific constant values are initialized.

3. Processing in Blocks: The padded message is divided into 512-bit blocks where each block is applied with a series of logical and bitwise operations.

4. Message Schedule: A schedule of 64 32-bit words from the 512-bit block is created.

5. Round Functions: For each block, 64 operations (in 64 rounds) are performed using logical and bitwise operations, including conditional functions, modular addition, and bitwise rotations.

6. Update Variables: The variables A, B, C, D, E, F, G, and H are updated after each round based on the round functions and the current block.

7. Output: The final values of A, B, C, D, E, F, G, and H are concatenated to get the 256-bit hash value.

1.3 Parallel Processing

Parallel processing is a method utilized to reduce the execution time of algorithms. **This paper presents the SHARAN-256 hashing algorithm, which employs parallel processing using a thread pool.** A thread pool reduces the overhead of frequently creating and destroying threads. Instead of creating a new thread every time a task needs to be executed, the thread pool maintains a pool of reusable threads. Using this method, the 512-bit chunks are computed in parallel using reusable threads.

2 Problem Statement

Several existing hashing algorithms, such as SHA (Secure Hash Algorithm), MD5 (Message Digest Algorithm 5), Blake, Whirlpool, among others, have been widely adopted for various cryptographic and security applications. In these algorithms vulnerabilities have been identified over time, making them susceptible to collision attacks where different inputs produce the same hash. In 1993, B. den

Boer and A. Bosselaers came across a kind of pseudo-collision for MD5 that had the same message with two different sets of initial values. SHA-256 also has some vulnerabilities as it has an attack for 46 (out of 64) steps of the compression function with practical complexity and preimage attacks on 41 steps of SHA-256.

The focus of this research is creating novel hashing technique that can stay ahead of potential vulnerabilities. The significance of developing and adopting new hashing methods is required to ensure the safety of digital systems, making it a pivotal aspect of the problem addressed in this paper.

3 Literature Survey

In [1], the authors discussed about the BlockChain, which is a decentralized database where nodes verify numerous data records in a public directory. This ensures transparency and eliminates the need for third-party control. This study proposes enhancing the RSA algorithm in BlockChain design to address challenges like memory consumption. It introduces a modified SHA-3 hash function for improved hashing speed, replacing operations with arithmetic operations. The results show reduced memory consumption and increased complexity, enhancing security and performance. Comparing the suggested algorithm to others shows that it works really well. It's a big step forward in BlockChain technology, making transactions safer, faster, and smoother.

In [2], the authors discuss about BlockChain technology, which can revolutionize information storage, task execution, and trust establishment among nodes with a focus on privacy and security. The cryptographic hash function secures blocks in a chain, ensuring data integrity. However, they are various challenges like time, resource constraints, and high memory usage persist. This study introduces "Hamming Bird 2" an algorithm creating a chaos key-based hash through logistic maps and lightweight cryptography. It is tested for time efficiency and resistance to cryptanalysis.So, the results show that it's really tough for hackers to break through using brute force attacks. This confirms that it's really good at making BlockChain safer and dealing with the usual problems with hashing.

In [3], the authors discusses about the SHA-2 algorithm, a upgraded version of SHA-1. It stands as a secure hash function resistant to various attacks like collisions, preimages, and second-preimages. Its security lies in a well-designed structure consisting a large message block size, numerous rounds and a complex round function.Even though it's known to be secure, it's important to use it carefully to prevent side attacks. Independent evaluations affirm SHA-2's security, but keeping in mind that no hash function is entirely immune. The paper takes a close look at SHA-2, pointing out what it's good at and where it could do better in terms of security or performance. It suggests ways to make it even more secure, considering that the challenges in keeping data safe are always changing and there is always a need to get better at it.

In [4], the authors proposed Multi-Level Hashing Algorithm (MLHA) for IoT devices that combines eight levels of modified SHA functions, allowing flexibility

in tailoring security levels based on device resources. MLHA enhances resistance to brute-force attacks, offers adaptability to specific IoT security needs, and demonstrates efficiency and energy savings. Tested on diverse devices and operating systems, MLHA proves secure, energy-efficient, portable, and versatile for widespread IoT applications.

In [5] the authors explains the current scenario of multimedia data transmission over the Internet, this study is concerned about time complexity and data security. Addressing the crucial aspect of data integrity, a novel SHA algorithm is introduced. This innovative solution enhances security with reduced execution time and improved bit difference values, offering a more robust approach to data transmission security.

In [6], the authors uses Secure Hash Algorithm (SHA) 2 and 3 for shuffling the exam questions in computer based tests to maintain integrity. Two methods were tested: one used specific digits of the hash value, and the other treated all digits equally. SHA-2 with the first method performed better in randomness and speed. They achieved the entropy of 3.88792 bits and execution time of 1.76585 seconds. Results show no significant difference in randomness between SHA-2 and SHA-3 but SHA-2 is faster in terms of computational efficiency.

In [7], the authors discusses that the hash functions are really important for things like security, cryptocurrencies and even small devices like medical tools. SHA-2 is a widely used type of hash function, and it's used in many different ways. People have worked a lot to make it faster and better, even creating special chips just for it. But because there are different needs for speed, resources, and energy use, it's hard to compare different ways of making SHA-2 work. This paper makes a kind of "testing ground" for SHA-2, where different ways of making it can be tried out and compared. They made it so that people can easily change things and see how it affects performance and energy use. By doing this, they can figure out the best ways to make SHA-2 work well for different situations.

In[8], the authors explores how secure hash algorithms (SHA-1 and SHA-2) work by analyzing their internal rounds. These algorithms generate hash codes and the randomness of these codes shows how secure they are. The paper uses tests to check the randomness of SHA-1 and SHA-2, and it suggests a better replacement for them that creates more random and secure hash codes. They tested everything using high-performance computers and a parallel computing platform called CUDA.

4 Methodology Adopted

4.1 Proposed Approach

1. Initialization: Eight 32-bit variables (h0 to h7) with predetermined values were initialized, the same as the original SHA-256.
2. Preprocessing: The input message is converted to a byte array using UTF-8 encoding. The original length of the message was calculated in bits.

3. Random Number Generation: A complex random number was generated based on the input message. This random number is generated using various mathematical operations like logs, trigonometric, factorial, square, and much more. The generated random number was appended to the message.

4. Padding: The message was padded with zeros until its length modulo 64 equals 56. This step ensures that the message length is congruent to 448 bits (mod 512), preparing it for chunk processing.

5. Chunk Division: The padded message was divided into 64-byte (512-bit) chunks for parallel processing.

6. Parallel Processing: Each chunk was processed independently using parallel processing techniques. Thread pools were employed to distribute and manage the processing of multiple chunks simultaneously which enhances computational speed.

7. Chunk Processing: For each chunk following operations were carried out:

   i) 16 words of 32-bit were extracted from each chunk to create the first 16 elements of 'w'(array of words of length 64).

   ii) The remaining 48 words (16-64) were generated by applying a loop and bitwise operations like Right Rotate and other logical operations to previously generated words.

   iii) On each word in 'w' a series of logical and arithmetic operations were performed based on the SHA-256 compression function.

   iv) The values of eight working variables ('a' to 'h') were updated iteratively based on the compression function's logic and the current word in 'w'.

8. Hash Result Compilation: The final values of 'h0' through 'h7' were compiled after processing all chunks to form the 256-bit hash result.

9. Output Formatting: Formation of the hash result as a 64-character hexadecimal string.

10. Return Hash: The formatted hash result was returned as the output of the proposed hashing algorithm.

4.2 Flowchart

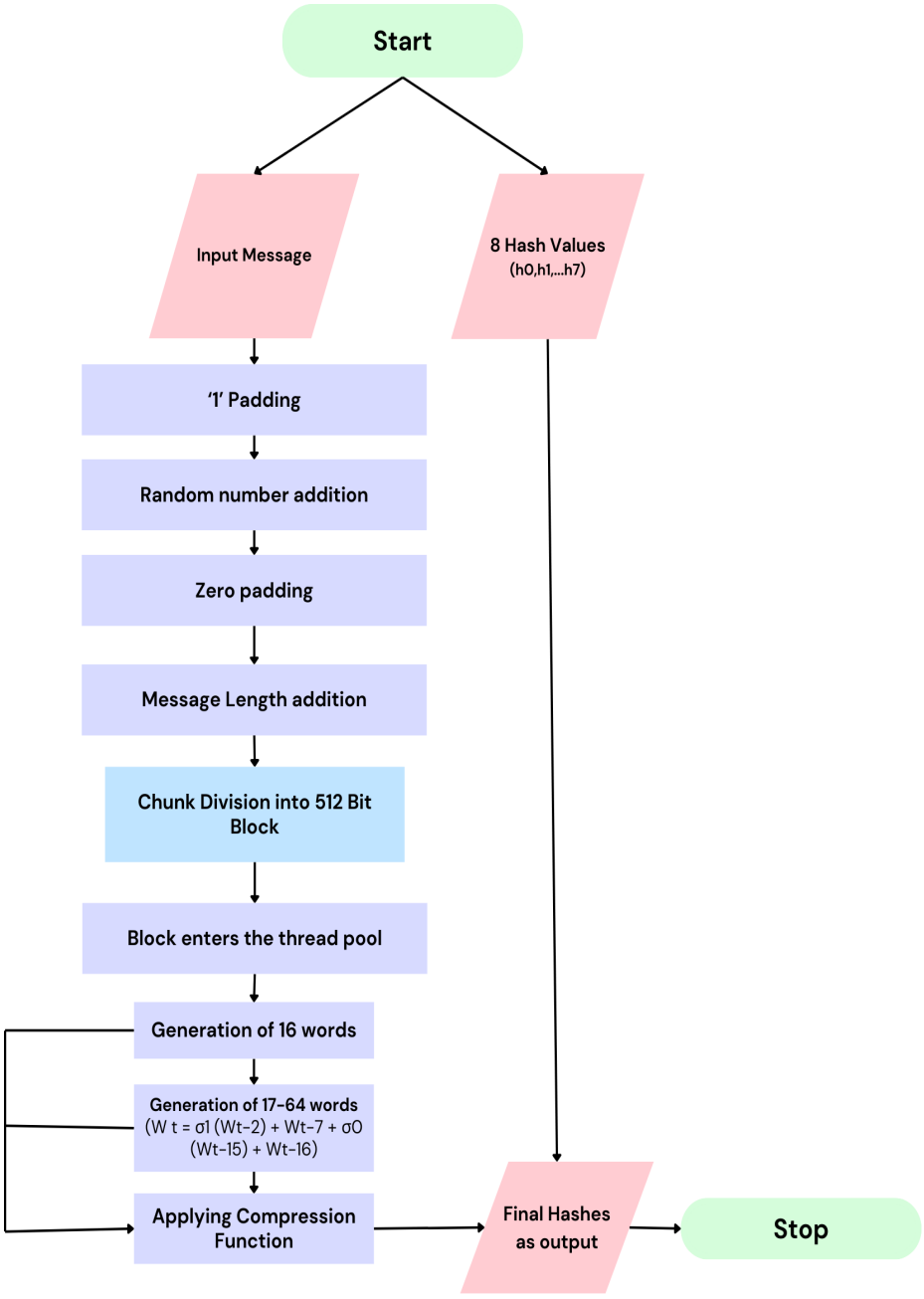

Fig. 1: Flowchart of the Proposed Hashing Algorithm

4.3 Algorithm Proposed

```
function SHARAN256(message):

    t0 = 0x6....667
    t1 = 0xB....E85
    t2 = 0x3....372
    t3 = 0xA....53A
    t4 = 0x5....27F
    t5 = 0x9....88C
    t6 = 0x1....9AB
    t7 = 0x5....D19

    k = [
        0x4....F98,
        0x7....491,
        0xB....BCF,
        0xE....BA5,
        ... 60 terms
    ]

    message_bytes = convert_message_to_bytes(message)
    random_value = generate_complex_random_number(message)
    append_random_value_to_message(message_bytes, random_value)
    padded_message = pad_message(message_bytes)
    chunks = break_into_chunks(padded_message)

    // Parallel processing step
    with ThreadPoolExecutor() as executor:
        futures = []
        for chunk in chunks:
            future = executor.submit(process_chunk, chunk, k, [t0, t1,
                t2, t3, t4, t5, t6, t7])
            futures.append(future)

        for future in futures:
            t0, t1, t2, t3, t4, t5, t6, t7 = future.result()

    hash_result = concatenate_hash_values(t0, t1, ..., t7)

    return hash_result
```

5 Results

5.1 Time Comparison of SHA-256 and SHARAN-256

To analyze the performance of the SHARAN-256 hashing algorithm against the SHA-256 algorithm, a time comparison test was conducted. The objective of

this test was to measure the execution time of both algorithms when hashing various test strings under identical conditions. Each algorithm was called 1000 times for hashing the same test string to ensure consistent and reliable results. While SHA-256 showed a slightly better performance, the difference in execution time between SHA-256 and SHARAN-256 was not much and is comparable. The additional security step in SHARAN-256, of calculating and padding a complex random number, contributed to a slight increase in execution time. Despite this added computational step, SHARAN-256 demonstrated its ability to maintain competitive performance while enhancing security measures.

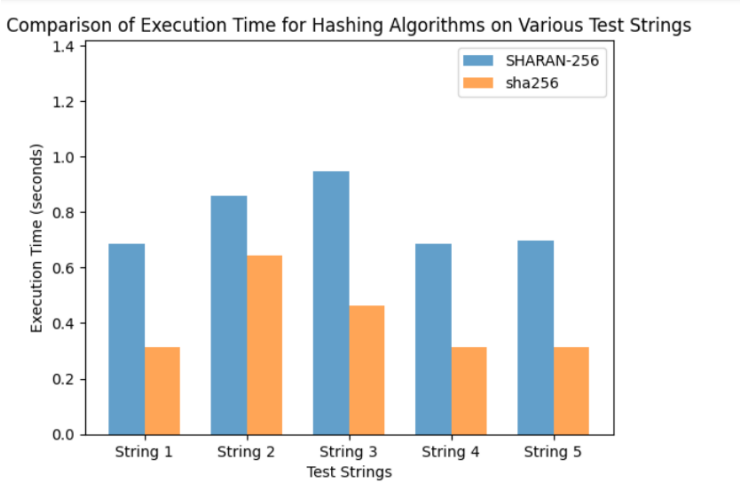

Fig. 2: Time comparison analysis between SHA-256 and SHARAN-256

5.2 Avalanche Test

The avalanche test is used to evaluate the quality of hashing function. It determines how sensitive the hashing function is to changes in input strings. The avalanche effect refers to the extent to which small changes in the input data lead to significant changes in the output (hash) value. Table 1 and 2 shows that just by changing 1 bit in input string the entire hash value differs in both SHA-256 and SHARAN-256.

Table 1: Avalanche Test for SHA-256

Message	**SHA-256 Hash**
“over”	5fb6a47e368e12e5d8b19280796e6a3d146fe391ed2e967d5f95c55bfb0f9c2f
“ouer”	b155e77c3e08dbe51143e8f26ee64dd58ecf7b2f0ebab69029c4a8d66141ce93
“oevr”	bc109eb7e2963cbeac331855df2392392406c703b72d2345aa23ac28f623a4ce
“oer”	8e08e27ab2dbdfb1d11e60ecb0cacd432994c43c8cbe77586aa5bbe4c280278a

Table 2: Avalanche Test for SHARAN-256

Message	**SHARAN-256 Hash**
“over”	3a95b7f2994dd1e6969b10fbde5d97fcc644d4edf51056c05bb2ba9efbb0630d
“ouer”	ca0781bb5a2776323c72b177ba362d291d5d0e2a3e418a8dfb98d9cf79bd1114
“oevr”	30ac3971910d0c675b0314ad18d1e398bc7ef1b7b865a37185c529e94f324488
“oer”	9768c641fee0e0342358fe59321d4e40cc8479e02f1264259c2fba5247498ed1

6 Conclusion

The SHARAN-256 algorithm demonstrates an enhancement in security through the integration of a random algorithm into the SHA-256 code giving it more randomness. Adding thread pools to distribute and manage the processing of multiple chunks simultaneously helped in making the algorithm time efficient. The proposed algorithm is tested by performing a Time evaluation test and Avalanche test. The Time Evaluation test compared the processing times of SHA-256 and SHARAN-256 algorithms, revealing that they were comparable. The Avalanche test demonstrated the robustness of SHARAN-256 by revealing significant changes in hash values with minor changes in the input string.

7 Future Work

The future work will be on optimizing and reducing the computational time of the SHARAN-256 algorithm by improving the random function so that it can handle large values input. Future work will also be done to thoroughly analyze the algorithm's performance under various scenarios and conduct a comprehensive set of tests to evaluate its performance, security, and efficiency such as differential cryptanalysis test, Scalability Test, Randomness Test, and many more.

8 References

1. Jasim, Aun H., and Ali H. Kashmar. "An Evaluation of RSA and a Modified SHA-3 for a New Design of BlockChain Technology." In Artificial Intelligence for Smart Healthcare, pp. 477-489. Cham: Springer International Publishing, 2023.
2. Ali, Alaa Abid Muslam Abid, Manar Joundy Hazar, Mohamed Mabrouk, and Mounir Zrigui. "Proposal of a Modified Hash Algorithm to Increase BlockChain Security." Procedia Computer Science 225 (2023): 3265-3275.
3. Sangheethaa, S., Arun Korath, and C. Ranjana. "Improvisation in SHA Algorithm." In 2023 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE), pp. 1-4. IEEE, 2023.
4. Sami, Teba Mohammed Ghazi, Subhi RM Zeebaree, and Sarkar Hasan Ahmed. "Designing a New Hashing Algorithm for Enhancing IoT Devices Security and Energy Management." International Journal of Intelligent Systems and Applications in Engineering 12, no. 4s (2024): 202-215.
5. Verma, Sandhya, and G. S. Prajapati. "Robustness and security enhancement of SHA with modified message digest and larger bit difference." In 2016 Symposium on Colossal Data Analysis and Networking (CDAN), pp. 1-5. IEEE, 2016.
6. Nugroho, Kuntoro Adi, Arimaz Hangga, and I. Made Sudana. "SHA-2 and SHA-3 based sequence randomization algorithm." In 2016 2nd International Conference on Science and Technology-Computer (ICST), pp. 150-154. IEEE, 2016.
7. Martino, Raffaele, and Alessandro Cilardo. "A flexible framework for exploring, evaluating, and comparing SHA-2 designs." IEEE Access 7 (2019): 72443-72456.
8. Al-Odat, Zeyad, Assad Abbas, and Samee U. Khan. "Randomness analyses of the secure hash algorithms, SHA-1, SHA-2 and modified SHA." In 2019 International Conference on Frontiers of Information Technology (FIT), pp. 316-3165. IEEE, 2019.