Damiana TELITI [1]
Olsi SHEHU [2]

Comparative Analysis of MD5, SHA-256, and SHA-512 Performance: Unexpected Cases of SHA-512 Outperforming SHA-256

Abstract

In the last several years, one of the concerns has been the resistance of hashing algorithms against supercomputers or quantum computers, which have higher computing capabilities compared to traditional computers. This study focuses on different hashing algorithms' performance under different inputs involved in ensuring data integrity. A Python script measures the performance of three different hashing algorithms in various type of inputs using the same hardware power. The first scenario uses the software to generate numeric inputs of varying lengths. Secondly, the same program generates different numbers of strings with different lengths, and lastly, it uses different text-based wordlists as an input. After each scenario, it assesses simultaneous or non-simultaneous time consumption, as are the efficiency metrics using factors such as speed, key length, hardware utilization, and robustness. Using the provided information, the designer could come up with measurable values for each of the features studied in this paper, as well as simulate attacks on quantum computers to see how well current hashing algorithms work with new hardware technologies. This study concludes that sha512 outperforms sha215 or md5 in certain scenarios, despite its more complex hashing algorithm and longer bit length.

Keywords: *hash, brute force, collision resistance, robustness, performance, quantum computer*

[1] MSc. Damiana TELITI, Computer Science Department, University College “Bedër”, Tirana, Albania.
[2] MSc. Olsi SHEHU, Computer Science Department, University College “Bedër”, Tirana, Albania.

1. Introduction

In the realm of data security, hash algorithms guarantee the integrity and validity of information. These functions are critical in guaranteeing data integrity verification and providing strong methods to detect any illegal modifications or tampering. However, with the sophistication of cyber-attacks and the advancement of computing technology, questions arise about the effectiveness of hash functions in maintaining data integrity. This research aims to explore the fundamental concepts of hash functions and their essential role in verifying data integrity. This paper aims to provide a thorough knowledge of the importance of hash functions in ensuring the integrity of data by extensively examining their theoretical foundations and practical uses. Moreover, this study seeks to enhance cryptographic security by creating a specific tool. This program aims to streamline the thorough examination of several cryptographic hash functions, specifically to evaluate their security effectiveness in defending against collision attacks. In the first scenario, the software generates numerical strings of different numbers of characters. For example, if the length of the generated string is five, it will produce numbers that range from 00000 to 99999. Subsequently, each value will undergo encryption using one of the chosen hashing algorithms. We measure and store in memory the encryption execution time and the number of encrypted values for each algorithm, repeat the same process for other hashing algorithms, and finally compare the results between them. The second scenario specifies the minimum and maximum lengths for each randomly generated string, as well as the total number of strings to generate. Unlike the preceding case, in which all strings have the same length and contain only numeric characters, the generated strings have different lengths and contain alphanumeric characters. Let's assume, for example, that the shortest string should contain 2 characters, the longest string should contain 10 characters, and the script should generate a total of 1000 random strings. The script will generate 1000 random alphanumeric strings with lengths ranging from 2 to 10 characters, encrypt each string, and produce a report detailing the elapsed time for encrypting these 1000 strings. The final phase will assess different hashing methods using wordlists containing text-based data. Each of the studied hashing algorithms encrypts each line of the same wordlist. We expect this method to execute faster than the two previous scenarios because it uses wordlists instead of generating strings. The script will create performance reports using the same methods as in the previous scenarios after it finishes running.

In order to guarantee "fair" competition between different hashing algorithms, it is necessary to run them concurrently and make use of an equivalent allocation of hardware resources [6]. In order to carry out multiple processes simultaneously, one can make use of the "multiprocessing" or "multithreading" modules in Python. In this situation, multiprocessing is more advantageous than multithreading [7] since it enables each process to execute in its own Python interpreter, effectively bypassing the Global Interpreter Lock (GIL) and therefore fully exploiting multiple CPU cores. This ensures that each algorithm utilizes the same hardware resources, enabling fair performance comparisons and precise benchmarking across concurrent processes.

2. Literature Review

In order to bolster security and mitigate vulnerabilities found in these hash functions, Roshdy et al. [1] conducted a study that presents a novel secure hash algorithm that combines features from SHA-256 and MD5. The suggested approach enhances complexity and improves resistance against sophisticated hash attacks by adjusting the message expansion in SHA-256 and implementing the double-Davies-Mayer scheme. This research reveals that the new algorithm provides superior security compared to SHA-256 and MD5. It successfully clears the avalanche and differential attack tests with greater chances. The suggested hash method produces a hash of 256 bits, which is well-suited for applications that need to ensure message integrity and use digital signatures. The architecture guarantees enhanced diffusion by executing an XOR operation between each output and the following input, preventing the occurrence of similar or the same outputs in the following stages. The new method's complexity and security are superior to those of SHA-256 and MD5. Further research could include a study on this algorithm's longer hash sizes.

Pittalia [2] focuses on the necessity of hashing methods in network security to guarantee the authenticity of data transferred between different entities. Almost every technology implements hash algorithms, including SHA-1, SHA-2, SHA-3, MD4, MD5, or any other, to produce safe data encryption. It can identify unwanted alterations to the input, authenticate data origins, and enable digital signatures. These algorithms should guarantee that the encrypted text remains unreversible and prevent collision attacks (two different inputs that result in the same hash). Furthermore, the author stresses the importance of hash algorithm implementation to protect data integrity and validate the integrity of sensitive information. The examination of MD5, SHA variants, and Whirlpool algorithms illuminates their distinct designs and uses in contemporary digital security.

The paper "A Flexible Framework for Exploring, Evaluating, and Comparing SHA-2 Designs" introduces a new workbench that aims to streamline the assessment and comparison of different SHA-2 hardware implementations. The framework offers a standardized environment to test different hash algorithms against common metrics such as performance, hardware usage, and energy consumption. The suggested platform ensures that comparisons between different algorithms are more uniform and focused on algorithms' architectural merits rather than hardware technologies. The workbench can accommodate a diverse range of design methodologies and effectively aid in the creation of novel SHA-2 transformation round cores. Designers achieve this by integrating their customized cores into the framework and effortlessly adjusting the surrounding circuitry. The platform's versatility allows for thorough analysis and improvement of architectural designs, ultimately aiding in the identification of the most efficient solutions that meet certain limits and criteria.

Martino et al. [3] conducted a study that presents a tool for the evaluation of different implementations of SHA-2 hash functions. The system is based on Hardware Description Language (HDL) and offers a variety of functionalities, including allowing users to thoroughly investigate various architectures and assess their performance and energy consumption. The process evaluates several optimization solutions, such as pipelining, loop unrolling, variable precomputation, data prefetching, and spatial reordering. This approach guarantees that optimization techniques largely impact outcomes, rather than variations in the target technology. The conclusions emphasize the workbench's usefulness in supporting detailed architectural analysis and optimization, enabling the creation of an efficient hashing algorithm. Future work will involve extending the framework to incorporate supplementary hash algorithms and cryptographic primitives. Furthermore, it will entail enhancing and investigating optimization approaches to improve the framework's ability to evaluate and compare various capabilities.

Roshdy et al. [4] conduct a comparative analysis of the performance and security characteristics of MD5 and SHA cryptographic hash algorithms. Developed in 1991, MD5 generates a 128-bit hash and is currently vulnerable to collision attacks, thereby compromising its security. The NSA designed the SHA hashing family to improve the security of the previous MD5 method by creating a new algorithm with a 256-bit bit length. It provides improved resistance against collision attacks and demonstrates greater processing performance. The process entails a meticulous examination

of their architecture, computational efficacy, and susceptibility to attacks. The study's findings indicate that SHA-256 performs at higher levels of security and efficiency in comparison to MD5. Future research suggests prioritizing the enhancement of the security and performance of different SHA algorithms and other cryptographic techniques.

Sumagita and Riadi [5] investigate the shortcomings of using MD5 for password encryption in online applications, highlighting its susceptibility to collision attacks. The study recommends using the SHA-512 algorithm to enhance security. The method conducts in-depth research on the relevant literature, has a profound understanding of the requirements, conducts a comprehensive vulnerability assessment, meticulously designs the system, and conducts rigorous testing. During the testing procedure, we conducted both penetration testing and user acceptability testing (UAT). The results of these tests demonstrated that the new system, based on SHA-512, provides a significant security improvement. According to the UAT findings, 86 percent of respondents had a favorable response, as indicated by the findings of the UAT conducted. The results of these tests demonstrated that the new system, based on SHA-512, significantly improves security. According to the UAT findings, 86 percent of respondents had a favorable response, as indicated by the findings of the UAT conducted. The current findings demonstrate that SHA-512 can provide a high level of competency and efficacy in online applications. Subsequent research will concentrate on enhancing cryptographic techniques in order to meet diverse security challenges that are continually changing.

Table 1 Overview of Related Works

Paper	Methodology	Conclusion	Recommendations for Future Research
Roshdy et al. [1]	This study applies the avalanche test, using a 1-bit difference and multiple-bit differences, to both a suggested algorithm and SHA256. We conducted the test on 20 different messages. It demonstrates that the suggested algorithm has a higher likelihood of passing the avalanche test compared to SHA256.	The suggested hash algorithm, which combines elements of MD5 and SHA-256, generates a 256-bit hash and exhibits greater intricacy and security compared to SHA-256 and MD5. The device has successfully passed testing for both avalanche and differential attacks, demonstrating its immunity to differential attacks.	Subsequent research should focus on increasing the hash size to improve security, as well as clarifying the method to improve performance and efficiency. Additionally, performing deep analysis to assess its resistance against new methods of attack and pursuing the establishment of formal standards will guarantee its dependability and wider adoption.

Pittalia [2]	A tabular comparison provides a concise overview of the block size, digest size, word size, number of rounds, collision resistance, and operations employed to evaluate the algorithms. The study assesses the security of each algorithm, taking into account known flaws, theoretical threats, and practical issues.	Effective implementation of hash algorithms is essential for ensuring cryptographic integrity and security. This study provides a comprehensive examination of the creation and analysis of MD5, several SHA variations, and Whirlpool. Employing suitable hash functions is critical for ensuring data integrity and verifying users' identities through digital certificates.	Future research ought to prioritize the development of hash algorithms that have improved security features, explore quantum-resistant alternatives, and boost the efficiency of existing algorithms, specifically for applications in blockchain and the Internet of Things (IoT).
Martino et al.[3]	The SHA-2 workbench is a highly efficient tool for investigating and enhancing various architectural options in SHA-2 hardware accelerators. It aids in determining the most suitable combinations of architectural strategies and target technologies, resulting in the optimization of hash rates and space efficiency.	This method involves incorporating a specific SHA-2 transformation round core into the framework, specifically designed for a particular FPGA. It repeatedly modifies the architectural parameters throughout the process. This technique ensures uniform and comparable outcomes by removing the impact of certain hardware technologies and synthesis tools.	Subsequent investigations should enhance the workbench to accommodate supplementary cryptographic algorithms and assess diverse hardware technologies. Expanding the foundation for software-programmable processors and creating user-friendly interfaces and automation tools will increase their application usability.
Roshdy et al. [4]	A literature study, a needs analysis, and the plan for the system are all components of the technique. The SHA-512 algorithm is the primary replacement for insecure encryption technologies. It evaluates the efficacy of the newly implemented encryption using user acceptance testing with questionnaires and penetration testing with brute force attacks.	According to the findings of the study, SHA-512 considerably improves password encryption especially applied in web applications. Penetration testing demonstrated the robustness of SHA-512 against attacks, and user acceptability testing revealed a high level of approval, verifying the procedure's efficiency in enhancing security.	Enhancing SHA-512 for improved performance and security should be the focus of future research. Future research should incorporate sophisticated cryptographic methods, regularly update algorithms, and conduct real-world tests to ensure ongoing security advancements.
Sumagita and Riadi [5]	An examination of the existing literature, a requirements analysis, and a vulnerability assessment of the system were all components of the study. The primary focus of this study was to evaluate the encryption capabilities of web-based application login systems. During the implementation phase, we upgraded the hash algorithm from MD5 to SHA 512. User acceptance testing and penetration testing subsequently took place.	Implementing SHA 512 significantly increased the security of web-based application login systems. This is evidenced by the success of penetration testing and the positive feedback from users, including 86% who agreed that the security had improved.	Further research should investigate the possibility of optimizing SHA 512 in order to cut down on processing time, as well as the possibility of integrating additional security measures to further strengthen web-based apps.

As observed in the previous table, multiple researchers have investigated the performance of various hashing algorithms. Each of these studies has its own features that differ from the others. The field of cryptographic hash functions has been the subject of previously conducted research that has thoroughly explored a variety of algorithms and the applications of such methods. Roshdy and his colleagues [1] proposed a new secure hash algorithm that combines the best parts of SHA-256 and MD5. This makes the system more complicated and harder to break with advanced hash attacks. Through the use of avalanche and differential attack testing, they were able to convincingly show the enhanced security of the implementation. In [2], Pittalia studied the importance of implementing hashing methods for network security. He demonstrated the implementation and importance of hashing algorithms in data protection, including MD5 and SHA. The creation of a

modular framework for researching SHA-2 architectures streamlined the evaluation and comparison of various SHA-2 hardware implementations. Martino et al. [3] introduced a tool for analyzing SHA-2 hash algorithms using HDL. Its primary focus was on strategies on how to optimize both performance and energy usage. Their research brought to light the significance of doing a detailed architecture analysis when it comes to optimizing hashing algorithms. In addition, Roshdy et al. [4] conducted a comparison study of the cryptographic hash algorithms, including MD5 and SHA. The study resulted in the superiority of SHA-256 over MD5 in terms of both security and efficiency. In conclusion, Sumagita and Riadi [5] conducted research on MD5's vulnerabilities in the context of password encryption and suggested SHA-512 as a more secure alternative. Their extensive testing, which included both penetration and user acceptability tests, indicated that they had made major enhancements to the security of the software.

This study relies on a multiprocessing method that uses the same amount of hardware resources. Each algorithm uses its own process and runs concurrently with other algorithms. This study tests MD5, SHA-256, and SHA-512 to examine their performance under various input types.

3. Hash Comparison with Python

Mathematically, hashing models transform a given string input into another string of a fixed length. They generate a different signature for each different input, guaranteeing that even a minor modification to the input will result in a completely different output. Information security, data integrity verification, password verification, and various other computer applications extensively implement hash algorithms to ensure unique identification and data protection [8].

Notable hash algorithms include MD5, SHA-1, SHA-256, and SHA-512. Modern technologies implement numerous hashing algorithms to guarantee the integrity and security of digital data. Each entity contains different attributes, including both its advantages and limitations. For example, certain methods demonstrate higher speed compared to other algorithms that prioritize security by increasing their mathematical complexity. Sometimes a hashing method is appropriate, but sometimes it has drawbacks. So, for a fair assessment between them, it is necessary to select the proper algorithm that best meets an application or scenario's needs and unique criteria. Conducting a fair comparison of different hashing algorithms is an essential step that can guarantee the credibility, integrity, and availability of digital data in a communication system [6].

Everyday applications frequently use hash algorithms to ensure data integrity and authenticate their identities. For instance, they verify if a file has undergone alteration during network

transmission. Hash algorithms, on the other hand, play a decisive role in password security, as they provide an effective means of protecting saved passwords by generating their hashes and storing them in an encrypted format. In addition to their primary function, hashing algorithms are inevitable in various areas of computer science, including network security, database integrity, and software validation [9]. This study investigates the advantages and limitations of md5, sha256, and sha512 by generating performance data. We use Python to create a tool that concurrently compares several hashing methods. We achieve this by either employing a predetermined collection of keywords or constructing a keyword list. This method accurately assesses the performance of each hashing algorithm. The objective of this study is to aid in understanding and choosing the most suitable hash algorithm for different usage circumstances.

4. Theoretical Background

We used the following Python packages to conduct this study:

1. The Hashlib package is one of the most advanced and reliable Python libraries. It provides a variety of hashing algorithms to generate hash-encrypted texts, as well as an intuitive and straight-forward implementation of the most popular hashing algorithms, including md5, sha256, and sha512. Technologies that depend on data verification, password storage, or various cryptographic procedures often include the hashlib module. With its user-friendly interface, this tool enables developers to easily incorporate different hashing processes into a project, making it an essential component in contemporary Python programming for jobs that involve secure data manipulation [10].

2. Time is a library that offers a wide range of functionalities for managing time-based processes. It provides different functionalities, including the ability to retrieve the current time, compute the execution time, and carry out operations related to time intervals. This module primarily uses a method known as time (). It provides the current time format and representations. The time module is necessary for applications that entail accurate time measurement, scheduling, or synchronization. The versatility and user-friendly nature of Python make it an essential tool for developers working on time-sensitive activities [11].

3. ArgParse is a powerful library that supports command-line argument parsing. It can provide an appealing interface by specifying the expected software arguments. Argparse enables users to specify arguments, set default values, and generate help messages automatically. The main features

of this module include the provision of positional and optional parameters, verification of data types, and the ability to personalize help explanations. It improves the flexibility and usability of Python scripts by making it easier to manage user input and configure program behavior using the command line [17].

4. Multiprocessing utilizes several processes running simultaneously; this robust library can significantly improve the performance of any software. Increasing the number of CPU cores increases the capacity for parallel processing. Multiprocessing enables parallelism by circumventing the Global Interpreter Lock (GIL), leading to enhanced efficiency. Process pool support enables simultaneous execution of a function on multiple input values. Differently from the multithreading Python module, multitasking shares the same amount of hardware resources for several parallel operations [6].

5. TQDM is an extremely efficient library for converting progress bars into loops and iterable processes. The interface provides a simple and intuitive platform for tracking the progress of tasks with a long execution time, which is particularly useful for activities such as data processing, file management, and computational processes. Software developers can effortlessly incorporate progress bars into their code using tqdm by encapsulating any iterable with the tqdm function. The module provides the ability to customize progress bars, allowing for the inclusion of additional details such as anticipated completion time and processing speed. Furthermore, it effortlessly combines with widely-used libraries such as Pandas and multiprocessing, hence improving its usefulness in diverse applications [13].

6. Subprocess is a versatile library that facilitates the creation of additional processes, establishes connections to their input, output, and error pipes, and retrieves their return codes. Python scripts provide a robust interface for executing and communicating with external programs and scripts. The subprocess module provides a more consistent and flexible way to replace earlier modules such as system and spawn. The primary operations provided by the subprocess module are subprocess. Run (), subprocess. Open (), and subprocess. Call (). These functions enable the execution of external commands, the transfer of data between processes, and the retrieval of execution results. The subprocess module is required for creating applications that must interact with the system shell or other programs. It enables the execution of complex shell commands and streamlines inter-process communication [14].

7. Random is a module that provides a wide range of functionalities for generating random

numbers of different datatypes, including integers, floating-point values, and sequences. It generates pseudo-random numbers and randomizes them. The main functionalities of the module include the randint () function, which generates random integers within a certain range; the random () function, which produces a random float between 0.0 and 1.0; and the choice () function, which selects a random element from a sequence that is not empty. Additionally, it provides functionality for rearranging sequences using the shuffle () method and producing random samples using the sample () method. The random module is essential for applications that include simulations, statistical sampling, games, security techniques, and any situation that requires the generation of random data [15].

8. The Matplotlib module offers a wide array of plotting features and is able to generate diverse charts, graphs, lines, plots, bars, histograms, and more. It supports the production of high-quality output in many formats and interactive settings on different platforms. It also has the ability to precisely modify the visual elements of the plot, such as adjusting the axes, labels, legends, and colors. Matplotlib is an indispensable tool in Python for data analysis, scientific research, and any application that requires accurate and informative visualizations of data [16].

4. Methodology

To conduct this study, a Python software is designed to evaluate the effectiveness of various hashing algorithms using the previously listed Python libraries. The tool will possess three main features.

On the first feature, the objective is to assess the efficiency of hashing algorithms by encrypting consecutive numbers of different lengths created by the "crunch" software. The purpose of this study is to evaluate how various algorithms perform against a given set of data that follows a known numeric pattern. It seeks to provide information about the efficiency of these algorithms when dealing with organized input.

Similar to the previous scenario, the second case it generates random alphanumeric sequences of varying lengths. The tool will assess the efficiency of hashing algorithms in handling input that is more unpredictable and diverse, mirroring real-world usage scenarios where data frequently doesn't have a consistent pattern.

In the last scenario, the tool will assess the efficiency of hashing algorithms by encrypting several wordlists consisting of text-based data. This is critical for applications such as password hashing

and dictionary-based attacks. In each situation, we will gather and examine significant performance data to determine the most efficient hashing method for unique circumstances.

Other essential features of the tool in this study include its ability to assess the efficiency of 20 different hashing algorithms, including SHA-512, SHAKE-128, RIPEMD-160, SM3, BLAKE2b, SHA-224, and others. It offers performance results for each algorithm. You can test hashing methods individually or concurrently. The evaluation then examines how many hashed keys it can generate in a given time frame.

Figure 1 Python Tool Manual

```
usage: hash.py [-h] [-f Text File | -n NUMERIC | -r RANDOM] [-l LINES] [-p] [-s]
               [-t {sha256,blake2s,sha3_512,shake_256,sha224,sm3,md5,sha3_384,sha384,md5-sha1,sha512_224,blake2b,sha3_256
,sha512,sha1,sha3_224,sha512_256,shake_128,ripemd160}]
               [-c COMPARE [COMPARE ...]]

This is a tool developed to test different hashing algorithms' performance.

options:
  -h, --help            show this help message and exit
  -f Text File, --file Text File
                        Select Text file to test hashes.
  -n NUMERIC, --numeric NUMERIC
                        Test hashes using numeric input. Example '-n 5' will generate hashes from 00000 to
                        99999.
  -r RANDOM, --random RANDOM
                        Test hashes using random strings. Example '-r 7' will generate hashes alphanumeric
                        strings with 7 caracters. If '-r 7,10' is added than the length of each string is
                        going to be random from 7 to 10 characters. To use this flag, -l flag is required.
  -l LINES, --lines LINES
                        Declare number or random strings to encrypt. Example: '-l 100' will generate 100
                        unique random strings.
  -p, --performance     Display Performance Graph
  -s, --stats           Display Performance Stats
  -t {sha256,blake2s,sha3_512,shake_256,sha224,sm3,md5,sha3_384,sha384,md5-sha1,sha512_224,blake2b,sha3_256,sha512,sha1,s
ha3_224,sha512_256,shake_128,ripemd160}, --type {sha256,blake2s,sha3_512,shake_256,sha224,sm3,md5,sha3_384,sha384,md5-sha
1,sha512_224,blake2b,sha3_256,sha512,sha1,sha3_224,sha512_256,shake_128,ripemd160}
                        Supported Hashes are: sha256 blake2s sha3_512 shake_256 sha224 sm3 md5 sha3_384
                        sha384 md5-sha1 sha512_224 blake2b sha3_256 sha512 sha1 sha3_224 sha512_256 shake_128
                        ripemd160
  -c COMPARE [COMPARE ...], --compare COMPARE [COMPARE ...]
                        Select 2 or more Hash Algorithms to compare as in following example: -c md5 sha256
                        sha512
```

Figure 2 displays the complete guide for the Python command-line utility. This manual offers comprehensive information regarding the functionalities of each flag, provides a complete list of supported hashing algorithms, and outlines the methods for performing commands to compare hashes simultaneously. Moreover, it offers instructions on selecting the particular hashes for testing.

Table 2 CPU Information

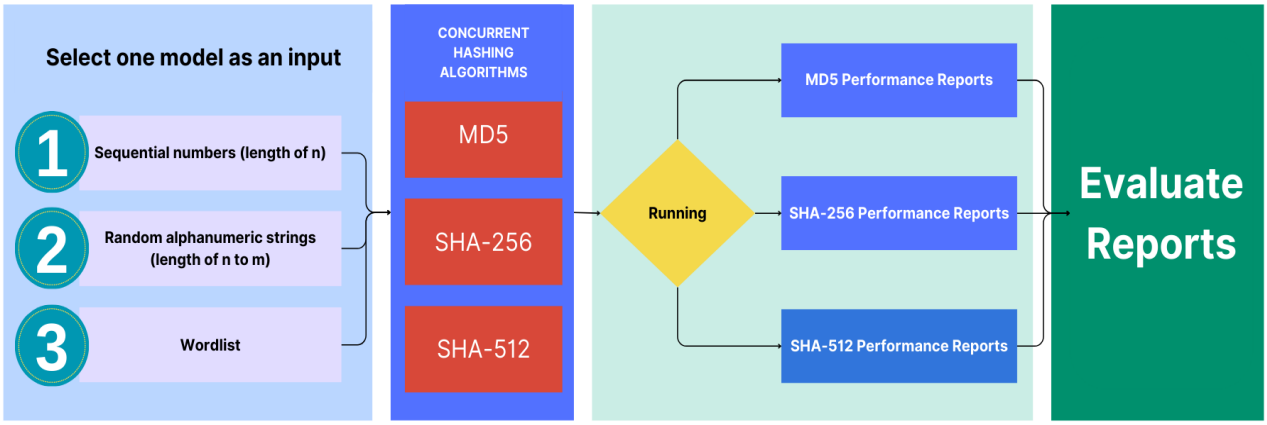
Product Collection	8th Generation Intel® Core™ i7 Processors
Total Cores	4
Total Threads	8
Max Turbo Frequency	4.20 GHz
Intel® Turbo Boost Technology 2.0 Frequency‡	4.20 GHz
Processor Base Frequency	1.90 GHz
Cache	8MB
Bus Speed	4 GT/s
TDP	15 W
Configurable TDP-up Base Frequency	2.10 GHz
Configurable TDP-up	25 W
Configurable TDP-down Base Frequency	800 MHz
Configurable TDP-down	10 W
Max Memory Size	32 GB
GPU Name	Intel® UHD Graphics 620

Regardless of the software component, the hardware parameters utilized for testing these hash algorithms will encompass crucial performance aspects. This includes the time required to generate the hash for different inputs. **Table 2** contains detailed information about the CPU used to perform this study.

4.1. Study workflow

The purpose of this research is to offer a full analysis of the performance of these algorithms in real-world circumstances. **Figure 1** illustrates the use of a Python tool to test these algorithms on generated ordered integers, random alphanumeric strings of varying lengths, and various wordlists. It's important to note that these three algorithms will run simultaneously, using the same hardware resource and under the same conditions.

Figure 2 Study Workflow



In the initial step, each hashing algorithm chooses an input model. The input model can consist of a sequential number represented as a string, a collection of randomly generated strings with different lengths, or wordlists obtained from the internet.

In the second stage, we select one of the following hashing algorithms: MD5, SHA-256, or SHA-512. Each of these algorithms will encrypt the input obtained from the first stage, either concurrently or independently.

Execution commences in the third phase. After finishing, performance reports are created. The reports encompass the overall duration of task completion, the frequency of iterations per second, and the count of false detections.

We study these reports to draw conclusions. The analysis aims to address multiple investigative questions: Is there a constant correlation between the hash method's bit length and its performance speed? What is the relationship between the complexity of an algorithm and its performance? Do the performances of the SHA algorithms significantly differ between variations within the same family? This research delves into these and other questions.

5. Results and discussion

In this section of work, we will test three different hash algorithms at different times. Testing hashes at different times, even with the same software and hardware, may yield different results due to the influence of software parameters at specific moments. For research purposes, we will conduct tests simultaneously and individually to make more comprehensive comparisons.

The hash algorithms we have chosen to analyze are MD5, SHA256, and SHA512. The following table shows the speed rate after testing algorithms in different conditions and at different times.

5.1. Comparison of hashes at different times

The following commands have provided the necessary data to populate the following table.

python hash.py -t [*hash type here*] -f [*wordlist directory here*] -s

python hash.py -t [*hash type here*] -n [*length of generated numbers*] -s

python hash.py -t md5 -r [*minimum length,maximum length*] -l [*number of generated strings*] -s

The developed Python tool operates as a command-line application. It performs different tasks and generates different results based on the arguments provided through different flags. **Figure 2** provides a comprehensive overview for a better comprehension of the preceding terminal

commands.

In summary: 1. The -t flag specifies the specific hashing algorithm for evaluation.

The -f parameter designates the text file for processing.

The "-s" flag presents performance metrics on the terminal.

The -n flag generates consecutive numbers with a specified length.

The -r flag produces random alphanumeric sequences.

The -l parameter indicates the exact count of created strings.

The -c parameter indicates the hashing algorithms to compare simultaneously.

Table 3 Separately comparison between hashes

	MD5(it/s)	**Time(s)**	**SHA256(it/s)**	**Time(s)**	**SHA512 (it/s)**	**Time(s)**
Small Wordlist (**8,930** combinations)	653,667.27	0.01	572,726.00	0.02	537,769.88	0.02
Big Wordlist (**2,303,940** combinations)	962,015.04	2.40	806,490.70	2.86	814,010.57	2.83
Numbers of **6 digits** (**1,000,000** combinations)	900,517.81	1.11	751,772.87	1.33	648,298.21	1.54
Numbers of **7 digits** (**10,000,000** combinations)	861,612.84	11.61	710,443.59	14.08	654,284.19	15.28
Numbers of **8 digits** (**100,000,000** combinations)	759,855.76	131.60	673,556.59	148.47	583,274.13	171.45
7-10 characters strings (**10,000** combinations)	12,947.11	0.77	10,041.73	**1.00**	12,552.01	**0.8**
10-20 characters strings (**10,000** combinations)	4,661.08	0.68	13,904.50	0.72	10,759.68	0.93
7-10 characters strings (**100,000** combinations)	1,255.38	79.66	1,248.67	80.09	1,153.25	86.71
10-20 characters strings (**100,000** combinations)	1,522.30	65.69	1,447.02	69.11	1,427.68	70.04

Table 3 demonstrates that the MD5 algorithm outperforms both SHA algorithms in every situation. However, MD5 is vulnerable to hash collisions because of its widely recognized limitations, including its relatively short hash length of 128 bits. On the other hand, even though SHA-512 has a bigger hash length of 512 bits and a more sophisticated hashing algorithm compared to SHA-256, there is no apparent difference in speed between the two.

The analysis of 10,000 different generated strings ranging in length from 7 to 10 characters led to an unexpected discovery. Even though SHA-512 has a more complicated algorithm, it demonstrated superior performance compared to SHA-256. However, the superior performance

was not evident when dealing with strings consisting of 10–20 characters with the same number of combinations. Therefore, we assume that SHA-512 performs better than SHA-256 at a specific character length threshold.

Executing processes at different times can lead to different hardware resource availability because other programs may use these resources completely or leave them entirely free. It can lead to inaccurate results. We must execute the algorithm under simultaneous processes to test different hashing algorithms under identical conditions, ensuring they have access to the same hardware resources.

5.2. Simultaneously comparison of hashes

The results of the aforementioned tests showed differences when testing three hashes at different time points, resulting in varying outcomes due to software parameters. In this section, we will explore how these three hashes will react when executed simultaneously, meaning concurrently at the same timeand using the same software.

```
python hash.py -c md5 sha256 sha512 -f [ wordlist directory] -p
python hash.py -c md5 sha256 sha512 -n [ length] -p
python hash.py -c md5 sha256 sha512 -r 7,10 -l 100000 -p
```

5.2.1. MD5, SHA256 and SHA512 tested under text-based file (wordlist)

The following table provides a concise overview of the hash rate and the time taken to test the algorithms on a substantial text-based file containing **2,303,940** lines. Since we are not utilizing the hardware resources to create new strings or integers, we expect a greater number of iterations per second in this situation.

Figure 3 Simultaneously wordlist

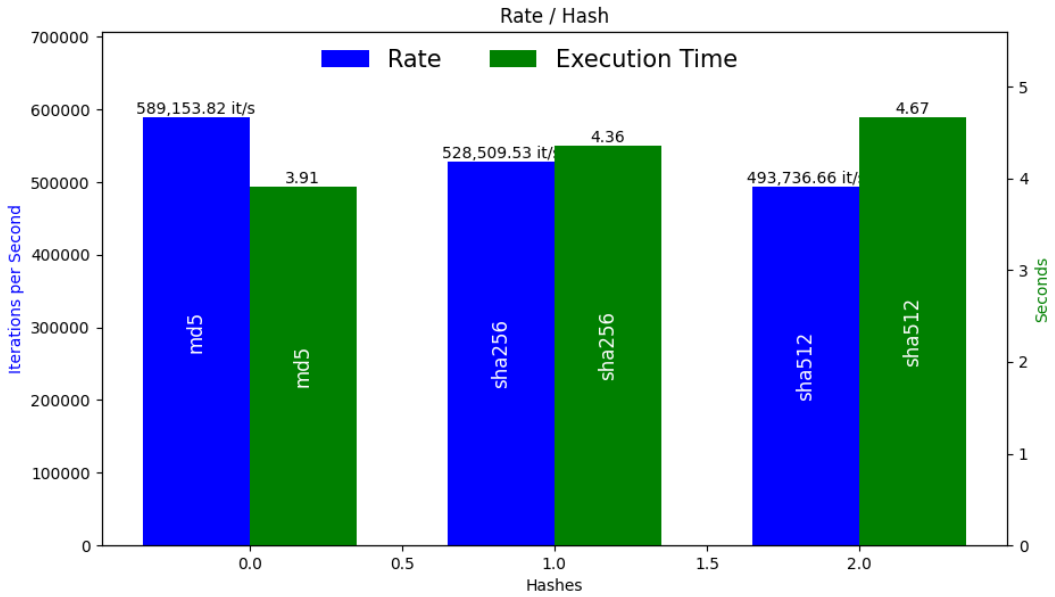


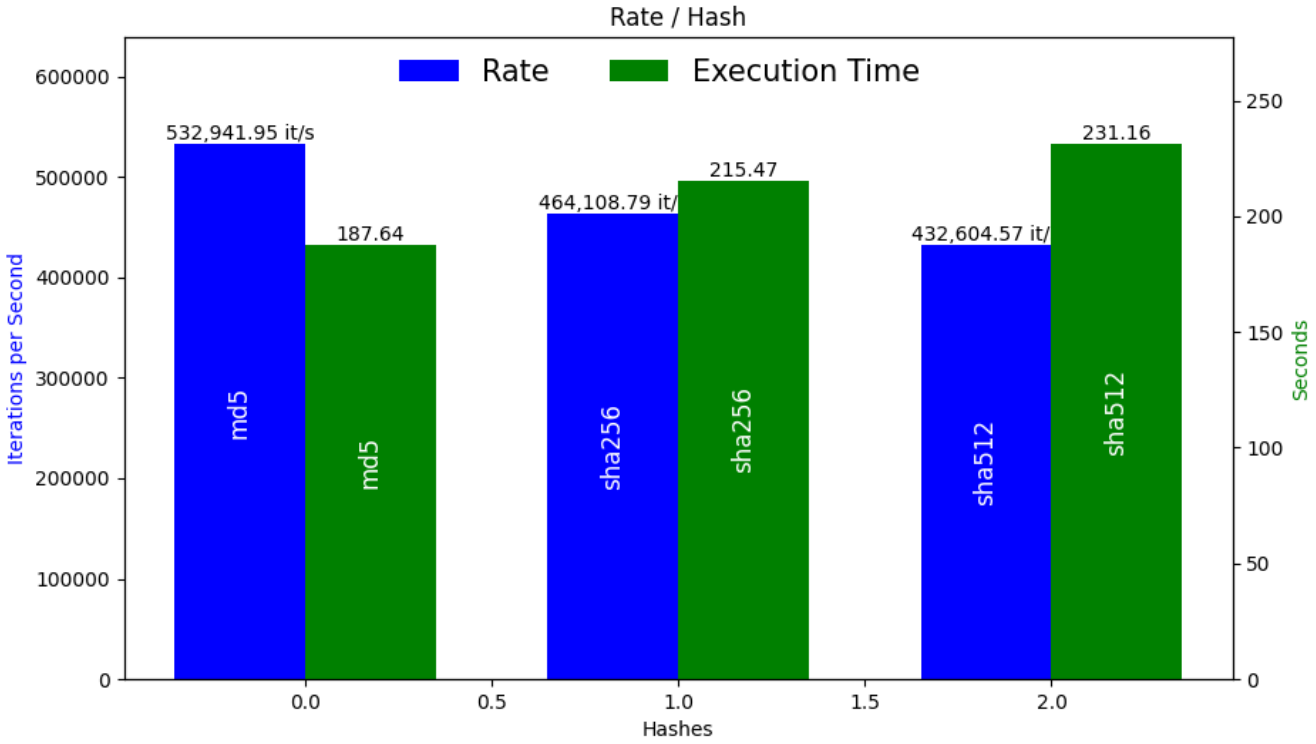
Figure 3 demonstrates that MD5 has the best pest performance compared to SHA-256 and SHA-512, mainly because of its shorter bit length and lesser complexity. It was able to successfully encrypt all of the wordlist lines in a time span of **3.91 seconds**.

In contrast, **SHA-256** demonstrated somewhat superior efficiency compared to **SHA-512**, encrypting all lines in **4.36** seconds, whereas SHA-512 completed the encryption in **4.67** seconds. It is worth mentioning that, although SHA-512 has a longer bit length and is more sophisticated, the difference in performance compared to SHA-256 is insignificant.

5.2.2. MD5, SHA256 and SHA512 tested under sequential number

The tables below provide a concise summary of the hash rate and elapsed time obtained from testing the algorithms using numbers created by the software. The numbers generated range from 00000000 to 99999999, resulting in a total of **100 million combinations**. In this situation, we expect a lower number of iterations per second because the CPU is responsible for both producing and encrypting the inputs.

Figure 4 Simultaneously number test



In **Figure 4**, **MD5** exhibits the highest level of performance, completing the encryption of all possible number combinations in **187.64** seconds, which is consistent with the prior example. The disparity in performance between the hashing algorithms is especially noticeable in this particular situation because of the larger number of inputs. The **SHA-256** algorithm successfully encrypted all possible combinations in **215.47** seconds, whereas the **SHA-512** algorithm required **231.36** seconds to complete the same task.

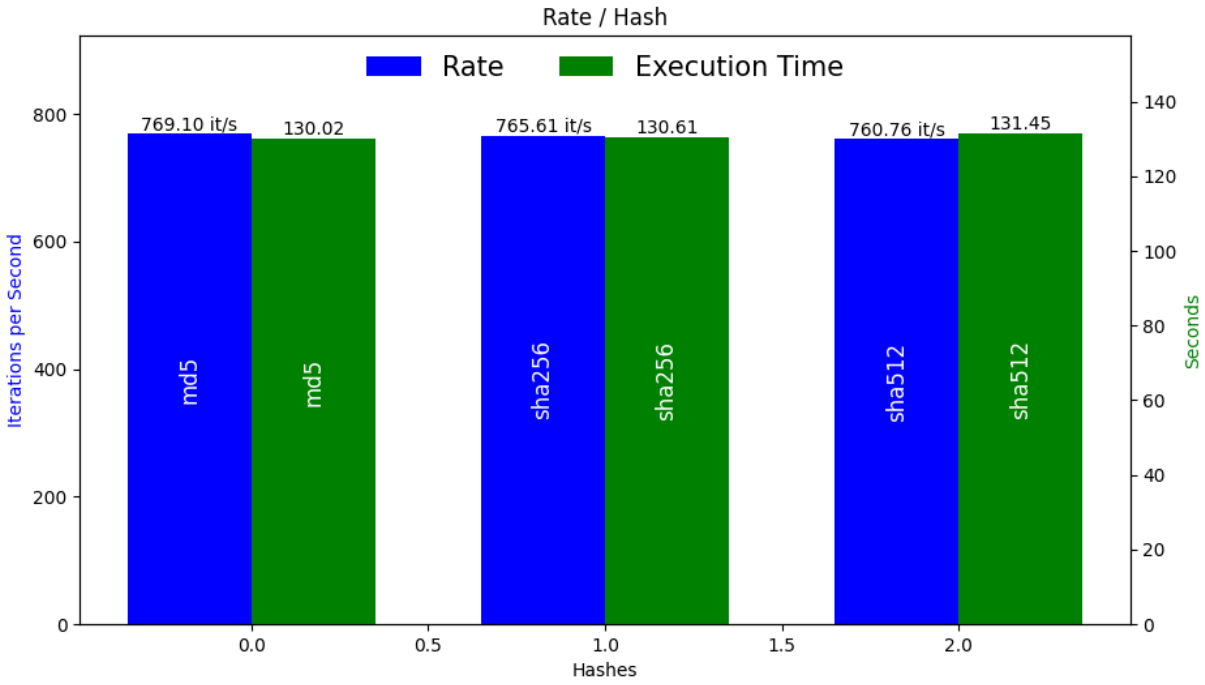
These results align with the initial expectations. We still need to determine whether SHA-512 will perform faster than SHA-256 when processing randomly generated strings, similar to the scenario where we compared the algorithms with different input quantities.

5.2.3. MD5, SHA256 and SHA512 tested under random strings

We conducted tests on the hashing algorithms using randomly generated strings that fell within a specific length range of 7 to 10 characters per key. In total, we evaluated 100,000 keys. Because we use the CPU for both key generation and encryption, we expect a reduced number of iterations per second compared to when working with wordlists, similar to the previous situation.

During the previous experiment, SHA-512 demonstrated the capability to surpass SHA-256 in terms of performance over various time intervals. It is still unclear whether SHA-512 will demonstrate the same exceptional performance in this scenario.

Figure 5 Simultaneously random strings first case



In **Figure 5**, it's showed that **SHA-256** method encrypts all randomly generated strings in **130.61** seconds, significantly faster than the **SHA-512** method, which takes **131.45** seconds. On the other hand, **MD5** outperformed both SHA algorithms, encrypting the same inputs in **130.2** seconds. Running these algorithms simultaneously yielded more consistent results than running them consecutively. It is important to note, however, that the performance of SHA-512 greatly improved while processing strings of unexpected lengths, whereas the performance of MD5 and SHA-256 was significantly worse when compared to other circumstances.

In the upcoming scenario, we will simultaneously test the three algorithms using the same amount of input but with longer string lengths. We will prepare for the upcoming scenario.

The hashing algorithm are tested with a random-length string within a specified boundary (10-20characters each key, 100,000 total keys). Results are as it is shown below

Figure 6 Simultaneously random strings second case

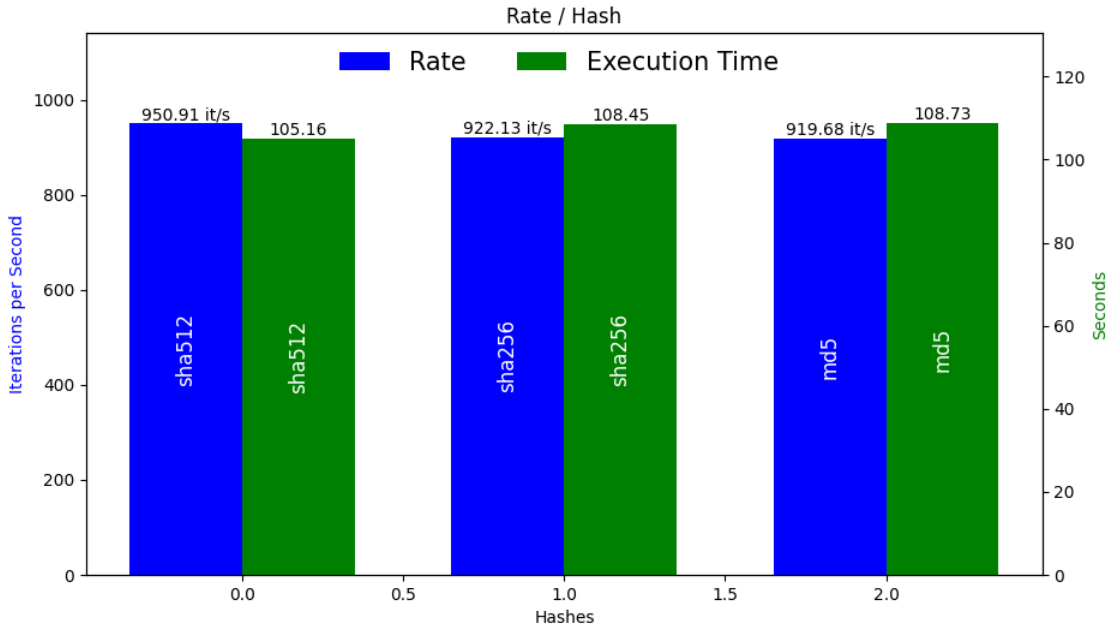


Figure 6 demonstrates that, in a manner that is analogous to the example presented earlier, SHA-512 demonstrates superior performance when dealing with inputs of varied lengths, despite the fact that it does not far beat the other algorithms. On the other hand, MD5 and SHA-256 both exhibit slower transmission times when given the same data. For the purpose of processing strings of varying lengths, the three hashing algorithms work with almost similar efficiency for processing strings of varying lengths, despite the fact that their bit lengths and algorithmic complexity are distinctively different from one another.

The table that follows summarizes the performance of each individual instance of simultaneous algorithm testing. This table displays the number of iterations per second as well as the amount of time that each algorithm requires to complete under a variety of different input conditions.

Table 4 Simultaneously Performance Report

	MD5(it/s)	**Time(s)**	**SHA256(it/s)**	**Time(s)**	**SHA512(it/s)**	**Time(s)**
Text Based (**2,303,940**Combination)	589,153.82	3.91	528,509.53	4.36	493,736.66	4.67
Numbers (**100,000,000** Combinations	532,941.95	187.64	464,108.79	215.47	432,604.57	231.16
7-10 Characters Strings (**100,000** Combinations)	790.10	130.02	765.61	130.61	760.76	131.45
10-20 Characters Strings (**100,000** Combinations)	950.91	105.16	922.13	108.45	919.68	108.73

Table 4 summarizes the performance of each individual instance of simultaneous algorithm testing. This table displays the number of iterations per second as well as the amount of time that each algorithm requires to complete under a variety of different input conditions. There is ample evidence to support the fact that SHA-256 typically exhibits greater speed than SHA-512 [18]. Occasionally, SHA-512 can demonstrate performance that is equivalent to or even surpasses that of SHA-256; however, this is uncommon. As indicated in Table 4, these instances commonly arise when the hashing algorithms handle randomly produced strings of different lengths. Situations where the algorithms encrypt predictable inputs, like sequential number inputs of specific lengths, do not exhibit this performance anomaly.

5.2.4. recommendations for future works

This study showcases the versatility of the Python testing tool since it is capable of supporting a wide range of algorithms, including but not limited to MD5, SHA-256, and SHA-512. This program has the capability to test more hashing algorithms, such as BLAKE2b, SHA-3-224, RIPEMD-160, and SHAKE-128, among others. In the future, researchers may use this capability to conduct comparable experiments on the different algorithms supported by the tool.

This study has revealed a vital discovery: the length of the input has a significant impact on the speed of encryption. Empirical evidence demonstrates that SHA-512 can surpass SHA-256 in terms of performance when applied to inputs of different lengths. Another option is a study to identify the exact length where SHA-512 performs faster than SHA-256.

Furthermore, a prospective field of research is utilizing the "crunch" software to produce varied wordlists from the inputs employed in this work, adhering to specified patterns. Pipelining in Linux can subsequently provide these wordlists to the testing algorithms. This approach would enable a more extensive evaluation of the algorithms' performance across a broader spectrum of input scenarios.

9. Conclusion

After conducting a thorough performance analysis, we have come to numerous important conclusions regarding the efficiency of the MD5, SHA-256, and SHA-512 hashing algorithms. MD5 regularly outperforms both SHA-256 and SHA-512 in a range of testing circumstances. The main reason why MD5 can handle data more quickly is because of its shorter bit length and lesser algorithmic complexity.

On the other hand, SHA-256 generally demonstrates somewhat superior efficiency compared to SHA-512. While SHA-256 may have a slight speed advantage, the difference in performance between these two algorithms is not significant. This implies that although SHA-256 may exhibit slightly better efficiency in certain situations, the higher bit length and intricacy of SHA-512 do not lead to a substantial improvement in performance compared to SHA-256.

Moreover, when handling inputs of different lengths, SHA-512 demonstrates better efficiency in comparison to SHA-256, although it still falls below MD5. In general, although MD5 offers the highest encryption speed, its lower bit length presents significant security vulnerabilities.

SHA-512 offers superior security compared to SHA-256 and MD5 because of its longer bit length and more sophisticated algorithm. However, its speed is only slightly slower than the other two algorithms. SHA-512 can outperform SHA-256 in some situations. Because of its strong security features, SHA-512 is well-suited for use in different applications or platforms, including cryptographic systems that require strong data integrity and secure communication protocols, or password storage systems. Thus, although SHA-512 may have slightly slower performance, its higher security properties make it an excellent option for applications that require high levels of security.

SHA-512 offers superior security compared to SHA-256 and MD5 because of its longer bit length and more sophisticated algorithm. However, its speed is only slightly slower than the other two algorithms. SHA-512 can outperform SHA-256 in some situations. Because of its strong security features, SHA-512 is well-suited for use in different applications or platforms, including cryptographic systems that require strong data integrity, secure communication protocols, and password storage systems. Thus, although SHA-512 may have slightly slower performance, its higher security properties make it an excellent option for applications that require high levels of security.

REFERENCES

[1] R. Roshdy, M. Fouad, M. Aboul-Dahab, (2013). Design and implementation a new security hash algorithm based on MD5 and SHA-256, *International Journal of Engineering Sciences & Emerging Technologies*, ISSN: 2231 – 6604

[2] Prashant P. Pittalia, (2019). A Comparative Study of Hash Algorithms in Cryptography, *International Journal of Computer Science and Mobile Computing*, ISSN 2320–088X

[3] Rafaele Martino, Alessandro Cilardo, (2019). A Flexible Framework for Exploring, Evaluating, and Comparing SHA-2 Designs.

[4] Surbhi Aggarwal, Neha Goyal, Kirti Aggarwa, (2014). A review of Comparative Study of MD5 and SHA Security Algorithm, *International Journal of Computer Applications* (0975 – 8887)

[5] Meiliana Sumagita, Imam Riad, (2018). Analysis of Secure Hash Algorithm (SHA) 512 for Encryption Process on Web Based Application, *International Journal of Cyber-Security and Digital Forensics* (IJCSDF) 7(4): 373-381

[6] Balkesen, C., Teubner, J., Alonso, G., & Özsu, M. T. (2013, April). Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)* (pp. 362-373). IEEE.

[7] Aziz, Z. A., Abdulqader, D. N., Sallow, A. B., & Omer, H. K. (2021). Python parallel processing and multiprocessing: A rivew. *Academic Journal of Nawroz University*, *10*(3), 345-354.

[8] Zheng, Y., Pieprzyk, J., & Seberry, J. (1993). HAVAL—a one-way hashing algorithm with variable length of output. In *Advances in Cryptology—AUSCRYPT'92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings 3* (pp. 81-104). Springer Berlin Heidelberg.

[9] Chi, L., & Zhu, X. (2017). Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (Csur)*, *50*(1), 1-36.

[10] Senthilselvi, A., Surya, H., & Raja, K. (2024, April). HaShuffle-Crafting Secure Passwords with a Splash of Shuffle Magic. In *2024 2nd International Conference on Networking and Communications (ICNWC)* (pp. 1-7). IEEE.

[11] Van Rossum, G. (2003). *An introduction to Python* (p. 115). F. L. Drake (Ed.). Bristol: Network Theory Ltd.

[12] Hellmann, D. (2011). *The Python standard library by example*. Addison-Wesley Professional.

[13] da Costa-Luis, C. O. (2019). tqdm: A fast, extensible progress meter for python and cli. *Journal of Open Source Software*, *4*(37), 1277.

[14] Langtangen, H. P. (2006). Basic Python. *Python Scripting for Computational Science*, 73-129.

[15] Korzeń, M., & Jaroszewicz, S. (2014). PaCAL: A Python package for arithmetic computations with random variables. *Journal of Statistical Software*, *57*, 1-34.

[16] Sial, A. H., Rashdi, S. Y. S., & Khan, A. H. (2021). Comparative analysis of data visualization libraries Matplotlib and Seaborn in Python. *International Journal*, *10*(1), 277-281.

[17] Youens-Clark, K. (2021). *Mastering python for bioinformatics*. " O'Reilly Media, Inc.".

[18] Gueron, S., Johnson, S., & Walker, J. (2011, April). SHA-512/256. In *2011 Eighth International Conference on Information Technology: New Generations* (pp. 354-358). IEEE.