Ju Ti : Indonesian Engineering Journal
ISSN : 2964-0490 (online)
Vol. 3, no. 01, 2024, pp. 24-29
https://jurnal.seaninstitute.or.id/index.php/juti

Merkle-Damgård as the Foundation of Hash Cryptography: A Study of Advantages and Limitations

Tri Ade Nia[1], Rudyanto Ompungsunggu[2], Arrant Ardi Sianipar[3], Jesimanta[4], Yoramo Waruwu[5]
[1,3]Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas Medan, Indonesia, [2]Jurusan Teknik Informatika, Universitas Katolik Santo Thomas Medan, Indonesia

Article Info

Article history:
Received, May 10, 2024
Revised, May 26, 2024
Accepted, Jun 15, 2024

Keywords:
Cryptography,
Merkle-Damgard,
Implementation,
Superiority,
Python.

ABSTRACT

The Merkle-Damgård algorithm is a key foundation in many cryptographic hash functions, including MD5, SHA-1, and SHA-256. This article discusses the working mechanism of the Merkle-Damgård structure, its advantages in maintaining data integrity, as well as the weaknesses that make it vulnerable to certain attacks. The study uses a quantitative approach through simulation and testing of Merkle-Damgård-based hash functions. This structure allows a variable-length input to be converted into a fixed-length hash output through a block-based iterative process and compression function. The results show that this algorithm successfully converts plaintext into a unique hash according to the hashing principle, as shown in both binary and ASCII representations. While effective in many digital security applications, this research highlights the importance of mitigating against potential attacks to improve the security of Merkle-Damgård-based algorithms.

Corresponding Author:
Tri Ade Nia,
Fakultas Ilmu Komputer,
Universitas Imelda Medan,
Jl. Bilal No. 52 Kelurahan Pulo Brayan Darat I Kecamatan Medan Timur, Medan - Sumatera Utara.
Email: corresp-author@gmail.com

1. INTRODUCTION

Hash cryptography plays a vital role in many aspects of digital security, including data integrity, authentication, and digital signatures.(Pardosi et al., 2023). One of the basic frameworks used in the design of cryptographic hash functions is the Merkle-Damgård construction, introduced by Ralph Merkle and Ivan Damgård in the late 1980s.(Akshima et al., 2022). With the increasing need for data protection in various systems, cryptographic algorithms are evolving to address threats to the integrity, confidentiality, and authentication of information.(Putra, nd). One of the most crucial components of cryptography is the hash function, which is used to ensure data integrity by generating a unique hash value from a digital message or document.(Haryono et al., nd).

Merkle-Damgård is one of the structures widely used in modern hashing algorithms such as SHA-1 and SHA-256.(Sitorus et al., 2024). This structure was introduced to address the problem of hashing with varying input lengths. With an iterative approach and the use of compression functions, Merkle-Damgård allows the hashing algorithm to produce output with a fixed length even though

the input data lengths vary.(Ghanimi et al., 2024). This structure also provides the avalanche effect property, where small changes to the input will produce very different hash values.

One of the main strengths of the Merkle-Damgård based algorithm is its ability to utilize padding and initialization values (Initial Vector/IV) to ensure the integrity of the hashing process.(Nishitha & Khare, 2025). This iterative process maintains the security of the hash value even though the data being processed is large and complex. In addition, the deterministic nature of this algorithm ensures that the same input will always produce the same hash, making it suitable for data verification and authentication functions.(Santoso et al., 2019).

However, despite its many advantages, the Merkle-Damgård algorithm is not free from security challenges. Attacks such as collision attacks and length extension attacks have become concerns in the implementation of this structure-based algorithm.(Arif & Nurokhman, 2023). Therefore, the purpose of this study will analyze the Merkle-Damgård method and implement and analyze its algorithm to understand its strengths and weaknesses in maintaining digital data security.

2. METHOD

This study uses a quantitative approach conducted through simulation and testing of Merkle-Damgård-based hash functions. Merkle-Damgård is a structure used in hashing algorithms, which allows the hash function to accept variable-length input and produce a fixed-length hash output. This structure utilizes a block-based iterative process with a compression function to maintain data security and integrity.

Figure 1 Merkle-Damgård flowchart

The Merkle-Damgard algorithm can be described mathematically as follows:

1. Start – The process starts.
2. Plaintext Input – The user enters the text to be hashed.
3. Padding – Adding padding bits so that the message length matches the required block size.
4. Compression Iteration – Messages that have been processed in blocks are processed iteratively using a compression function, where each output depends on the results of the previous iteration.
5. Final Result – The final hash value is obtained after all blocks are processed.
6. End – The hashing process is complete.

This diagram reflects the basic principles of the Merkle-Damgård construction, which is used in many hash functions such as MD5, SHA-1, and SHA-2.

3. RESULTS AND DISCUSSION

Plaintext Representation

The first step in hashing is to convert the plaintext into binary form using ASCII codes. In this example, the plaintext "QUE" is used, which is converted as follows:

Table 1 Plaintext Representation

Character	Ascii	Binary
Q	81	01010001
U	85	01010101
E	69	01000101

After conversion, the final result is: QUE → 01010001 01010101 01000101

Block Distribution

Merkle-Damgård processes data in blocks of fixed size. In this example, the block size is 8-bit. Since the message already has the appropriate length (24 bits), the block division is done as follows:

Table 2 Block Distribution

Block	Data (Binary)	ASCII Characters
M_1	01010001	Q
M_2	01010101	U
M_3	01000101	E

Padding

Padding is required if the message length does not fit within the specified block size. However, in this example, the message length is already within the specified block size (24 bits), so no padding is required.

Initialization of Initial Values (IV)

Each hashing process uses an initial value (IV) before the compression iteration is performed. In this example, the initial value is simplified to:

H_0 = 00000000 (8 bits)

Compression Function

In Merkle-Damgård, a compression function is used to process each block sequentially. In this example, the compression function used is the XOR (⊕) operation between the previous hash value and the current message block.

Compression function formula:

H_i = H_{i-1} \oplus M_i]

Compression Iteration

After the initial value (IV) is determined, the compression iteration is performed as follows:

Table 3 Compression Iteration

Iteration	H Previous	Block (M_i)	XOR Result (H_i)
1	00000000 (H_0)	01010001 (M_1)	01010001 (H_1)
2	01010001 (H_1)	01010101 (M_2)	00000100 (H_2)
3	00000100 (H_2)	01000101 (M_3)	01000001 (H_3)

Each result of an iteration is used as input for the next iteration, until a final hash value is obtained.

Final Result (Hash Value)

After the compression iteration is complete, the final result is obtained as follows:

a. Hash in binary: 01000001
b. Hash in ASCII: 'A'

These results show that the Merkle-Damgård construction produces a unique hash value for any given input, depending on the compression function used.

Process Summary

The following is a table that represents the iteration process of the Merkle-Damgard algorithm for the plaintext "QUE":

Table 4 Merkle-Damgard algorithm iteration

Iteration	(H_{i-1})	(M_i)	(H_i = H_{i-1} opplus M_i)
1	00000000	01010001 (Q)	01010001
2	01010001	01010101 (U)	00000100
3	00000100	01000101 (E)	01000001

Colum Description:

1. Iteration: The (i)th step in the iteration process.
2. (H_{i-1}): The hash value of the previous iteration.
3. (M_i): The (i)th message block currently being processed.
4. (H_i): Temporary hash value resulting from the XOR operation ((oplus)) between (H_{i-1}) and (M_i).

The final hash value ((H_3)) is 01000001, which in ASCII is the character 'A'.

Final Hash: `01000001` (A)

a. This example uses a simple compression function (XOR) to make it easier to understand. In real implementations, compression functions are much more complex and involve powerful mathematical operations.
b. The Merkle-Damgard algorithm is typically used with larger block sizes (e.g., 512 bits) and more sophisticated compression functions (such as SHA-256).
c. Padding and initial values (IV) are also more complex in real implementations.

With this example, you can see how the Merkle-Damgard algorithm iteratively processes a message to produce a hash value.

Implementation

The final result:

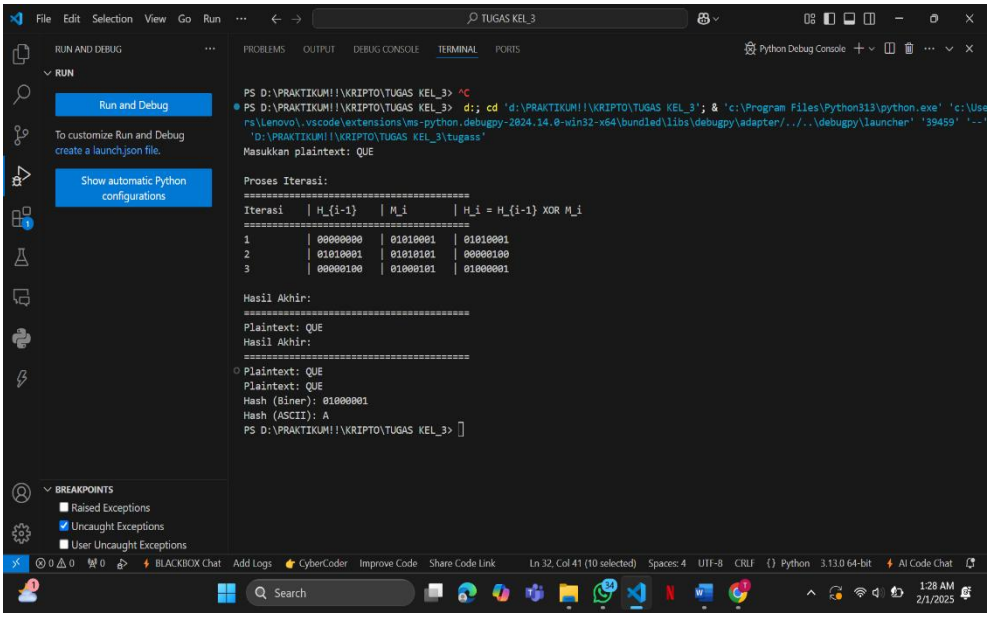

Figure 2 XOR based hash

A simple implementation of an XOR-based hash, which works by taking each character of the plaintext, converting it to binary, and repeatedly performing the XOR operation.

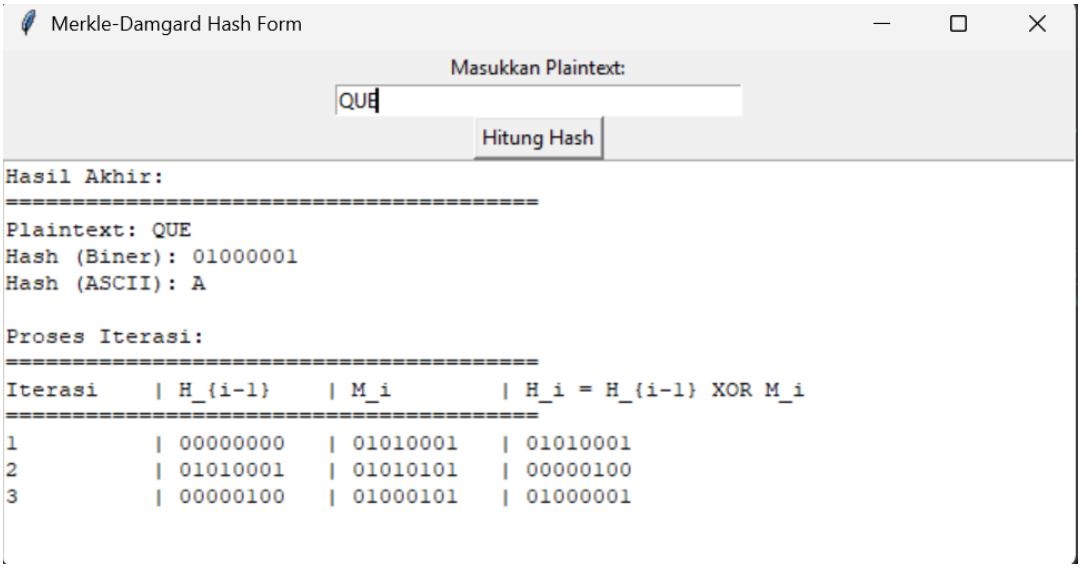

Figure 3. Plaintext Input

Plaintext Input

The plaintext entered is "QUE", which is 3 characters long. Each character will be converted into its 8-bit binary representation:

a. Q = `01010001`
b. U = `01010101`
c. E = `01000101`

The following is a table that summarizes the hashing iteration process using the Merkle-Damgård algorithm with the XOR operation:

Table 5 Merkle-Damgård Iteration Process

Iteration	Previous Hash (Hi−1H_{i-1}Hi−1)	Message Block (MiM_iMi)	XOR operation	Hash Result (HiH_iHi)
1	00000000	01010001	00000000 ⊕ 01010001	01010001
2	01010001	01010101	01010001 ⊕ 01010101	00000100
3	00000100	01000101	00000100 ⊕ 01000101	01000001

The final result

After all iterations are completed, the final hash value obtained is:

a. Hash (Binary): 01000001
b. Hash (ASCII): A

After going through a series of iterations with XOR operations between the previous hash value and the message block, the final hash value obtained in binary form is 01000001. When converted into ASCII representation, this binary value corresponds to the character "A" in the ASCII table. This shows that the algorithm has successfully converted the plaintext into a unique hash output according to the hashing principle.

4. CONCLUSION

Merkle-Damgård has been an important foundation in the development of cryptographic hash functions. By understanding its basic theory, formulas, and manual implementations, we can better appreciate the importance of this structure in digital security applications. However, there is a need for mitigation to deal with various attacks that can weaken Merkle-Damgård-based algorithms.

REFERENCES

1. Akshima, Guo, S., & Liu, Q. (2022). Time-Space Lower Bounds for Finding Collisions in Merkle-Damgård Hash Functions. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *13509 LNCS*, 192–221. https://doi.org/10.1007/978-3-031-15982-4_7
2. Arif, Z., & Nurokhman, A. (2023). Analisis Perbandingan Algoritma Kriptografi Simetris Dan Asimetris Dalam Meningkatkan Keamanan Sistem Informasi Comparative Analysis of Symmetric and Asymmetric Cryptographic Algorithms in Improving Information System Security. In *JTSI* (Vol. 4, Issue 2).
3. Ghanimi, H. M. A., Melgarejo-Bolivar, R. P., Tumi-Figueroa, A., Ray, S., Dadheech, P., & Sengan, S. (2024). Merkle-Damgård hash functions and blockchains: Securing electronic health records. *Journal of Discrete Mathematical Sciences and Cryptography*, *27*(2), 237–248. https://doi.org/10.47974/JDMSC-1878
4. Haryono, W., Kom, S., & Kom, M. (n.d.). *TEORI KRIPTOGRAFI DAN APLIKASI PENERBIT CV.EUREKA MEDIA AKSARA*.
5. Nishitha, T., & Khare, A. (2025). Smart Contract-Enhanced Residual GRU with Merkle-Damgård Cryptography for IoT Attack Detection. *Engineering, Technology and Applied Science Research*, *15*(1), 19331–19336. https://doi.org/10.48084/etasr.8860
6. Putra, A. E. (n.d.). *FUNGSI HASH PADA KRIPTOGRAFI*.
7. Santoso, M. H., Girsang, N. D., Siagian, H., Wahyudi, A., & Sitorus, B. A. (2019). Perbandingan Algoritma Kriptografi Hash MD5 dan SHA-1. In *Prosiding Seminar Nasional Teknologi Informatika* (Vol. 2).
8. *Sistem Keamanan Informasi*. (n.d.).
9. Sitorus, N., Sharon, J., Sinaga, G., Samosir, S. L., Terapan, S., Rekayasa, T., Lunak, P., & Del, I. T. (2024). Analisis Kinerja Algoritma Hash pada Keamanan Data: Perbandingan Antara SHA-256, SHA-3, dan Blake2. *Jurnal Quancom*, *2*(2).
10. Prasetyo, B., & Hidayat, R. (2019). Implementasi Algoritma MD5 untuk Integritas Data pada Sistem Keamanan Informasi. Jurnal Sistem Informasi dan Teknologi, 7(3), 123-130. (n.d.).
11. Rahardjo, B., & Susanto, A. (2017). Penerapan Fungsi Hash dalam Pengamanan Data Transaksi Elektronik. Jurnal Informatika dan Komputer, 10(2), 89-97. (n.d.).
12. Saputra, A., & Wijaya, D. (2020). Studi Komparasi Algoritma Kriptografi Hash SHA-1, SHA-256, dan MD5. Jurnal Teknologi dan Sistem Informasi, 5(1), 78-85. (n.d.).