Login : Jurnal Teknologi Komputer
ISSN : 2302-9692 (print) | 2723-8695 (online)
Vol. 18, No. 02, 2024, pp. 127-132
LOGIN http://login.seaninstitute.org/index.php/Login

Analysis of Cryptographic Utilization with Merkle-Damgård Algorithm

Peronika Ulianti Nainggolan[1], Dennis Afrilyans Manik[2], Laurenzio Gratian A. Daeli[3], Flory E Bako[4]
Teknik Informatika, Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas

Article Info

Keywords:

Cryptography, Merkle-Damgård, Hash Functions, Data Security, Digital Signatures

ABSTRACT

Cryptography plays an important role in data security, especially in authentication and digital signatures. One method that is widely used in cryptographic hash functions is the Merkle-Damgård algorithm. This algorithm allows the transformation of variable-sized data into a fixed hash value through an iterative process with a compression function. This study aims to analyze and understand the working mechanism of the Merkle-Damgård algorithm and its implementation in a data security system. The methods used in this study include the hashing process by dividing messages into fixed blocks, adding padding, initializing the initial value, and iterating the compression function. Testing was carried out with the example of the plaintext "ABC" using a simple XOR operation. The results of the analysis show that the Merkle-Damgård algorithm can produce unique and deterministic hash values, making it effective in detecting data changes. However, this algorithm also has weaknesses against collision attacks, which is a challenge in its development. Therefore, a deep understanding of this algorithm is essential in improving the security of modern hash functions.

Corresponding Author:

Peronika Ulianti Nainggolan
Teknik Informatika, Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas

1. INTRODUCTION

In cryptography, especially digital signature applications, hash functions play a crucial role in ensuring data security, especially in authentication, digital signatures, and password storage.(Patricia, 2024). One of the basic structures in the formation of a hash function is the Merkle-Damgård algorithm, which is the foundation for various popular hash algorithms such as MD5, SHA-1, and several variants of SHA-2.(Fajrin, 2023).

In the application context, Merkle-Damgård is used in various hash algorithms such as MD5 and SHA. Research shows that although this method is effective, it has a weakness related to padding which can cause two different strings to be identical after the padding process. This poses a challenge in maintaining the uniqueness of the output of the hash function.(Ariesanda, nd).

Further analysis shows that the implementation of Merkle-Damgård can affect data security, especially in terms of data integrity and authenticity. This study emphasizes the importance of evaluating the effects caused by the cryptographic process to ensure that the initial purpose of using cryptography as a security technology is still achieved.(Indrayani et al., 2024).

The Merkle-Damgård algorithm allows a hash function to convert variable-sized data into a fixed-length output, while ensuring that small changes to the input data will produce very different hash values. This makes it an effective tool in detecting changes or manipulation of data. However,

behind its advantages, this algorithm also has weaknesses that can be exploited by irresponsible parties, so understanding how this algorithm works and its security is very important.

Therefore, this report is prepared to discuss the basic concept of the Merkle-Damgård algorithm, how it works, and examples of its implementation in cryptographic hash functions. Through this discussion, it is expected to provide a deeper understanding of the role and challenges of the Merkle-Damgård algorithm in the world of digital data security.

2. METHOD

The Merkle-Damgård algorithm is a construction method used to build a cryptographic hash function from smaller compression functions. This algorithm is the basis for many popular hash functions, such as MD5, SHA-1, and SHA-256.(Sitorus et al., 2024). The Merkle-Damgård construction works by processing messages in fixed blocks using a compression function in an iterative manner.(Tiwari, 2017). This allows efficient processing of large data sizes and ensures deterministic properties of the hash function.

Working Principle of the Merkle-Damgård Algorithm

Merkle-Damgård works with the following steps:

a. Message Sharing
The input message M is divided into t blocks of fixed size, say 512 bits per block:
$$M=(M1,M2,...,Mt)M = (M\_1, M\_2, ..., M\_t)$$

b. Padding
If the message length does not fit into a multiple of the block size, padding is added to ensure that it does. Padding usually consists of a single "1" bit followed by several "0" bits until the total length reaches a multiple of the block. In addition, the original length of the message is also included in the padding to increase security.

c. Initialize Initial Values
The hashing process starts with a fixed initial value H_0, which is predetermined by the hash algorithm used.

d. Compression Function Iteration
Each message block MiM_i is processed iteratively using a compression function f, which combines the current block with the hash value of the previous iteration:
$$[H\_i = f(H\_{i-1}, M\_i)]$$
Where:
(H_i): Temporary hash value at iteration (i).
(H_{i-1}): The hash value of the previous iteration.
(M_i): The (i)th message block.
(f): Compression function.

e. The final result
Afterall blocks are processed, the temporary hash value of the last iteration (H_t) becomes the final hash output. [text{Hash}(M) = H_t]

3. RESULTS AND DISCUSSION

Here is an example of a cryptographic solution using the Merkle-Damgård algorithm with the plaintext "ABC". For simplicity, we will use a simple compression function and simplified initial values. The purpose of this example is to illustrate the iterative process of the Merkle-Damgård algorithm.

a. Step 1 Plaintext Representation
The plaintext used is "ABC".

Table 1 Convert each character into binary form based on its ASCII code:

character	ASCII	binary
A	65	01000001
B	66	01000010
C	67	01000011

Combine the binary representations of the plaintext: ABC = 01000001 01000010 01000011

b. Step 2 Block Distribution
The block size used is 8 bits per block. Each character in the plaintext is converted into a separate block:
   a. Block 1 (M_1) = 01000001 (A)
   b. Block 2 (M_2) = 01000010 (B)
   c. Block 3 (M_3) = 01000011 (C)

c. Step 3 Padding
The total length of the message is 24 bits, which is a multiple of 8-bit blocks. Therefore, no padding is required.

d. Step 4 Initialize Initial Values
The initial value (H_0) used is 00000000 (8 bits).

e. Step 5 Compression Function
The compression function used is an XOR operation between the previous hash value and the current message block:
Hi = Hi−1⊕ MiH_i = H_{i-1} \oplus M_iHi=Hi−1⊕Mi

f. Step 6 Compression Iteration
   1. Iteration 1:
      H_0 = 00000000
      M_1 = 01000001
      $H_1 = H_0 \oplus M_1$ = 00000000⊕01000001 = 01000001
   2. Iteration 2:
      H_1 = 01000001
      M_2 = 01000010
      $H_2 = H_1 \oplus M_2$ = 01000001⊕01000010 = 00000011
   3. Iteration 3:
      H_2 = 00000011
      M_3 = 01000011
      $H_3 = H_2 \oplus M_3$ = 00000011⊕01000011 = 01000000

g. Step 7 Output Hash
The final hash value (H_3) after all iterations is 01000000. In ASCII, this value represents the character "@".The following is a table that represents the iteration process of the Merkle-Damgard algorithm for the plaintext “ABC”:

Iteration	(H_{i-1})	(M_i)	(H_i = H_{i-1}⊕ M_i)
1	00000000	01000001 (A)	01000001
2	01000001	01000010 (B)	00000011
3	00000011	01000011 (C)	01000000

Column 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 01000000, which in ASCII is the '@' character.
   Final Hash: 01000000 (@)

Testing with Python

Shows the program interface for hashing with the Merkle-Damgård algorithm. The entered text ("UNIKA") is converted into binary format and displayed as a bit representation. The binary text is then divided into blocks for further hashing.

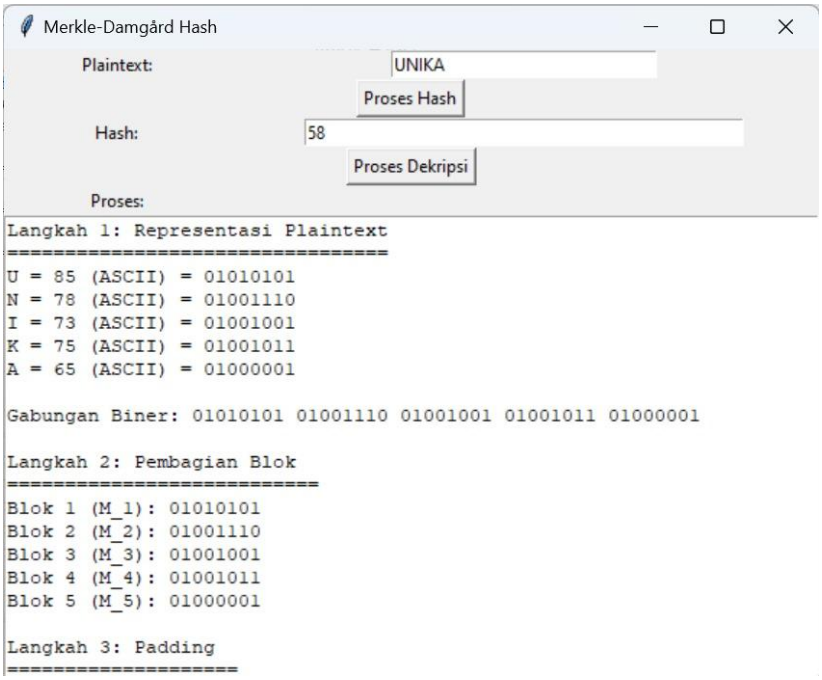

Figure 1. hashing process

The program displays the results of the block division and shows that no padding is needed because the data length is already in accordance with the block size. The initial value is initialized as the basis for subsequent hashing calculations.

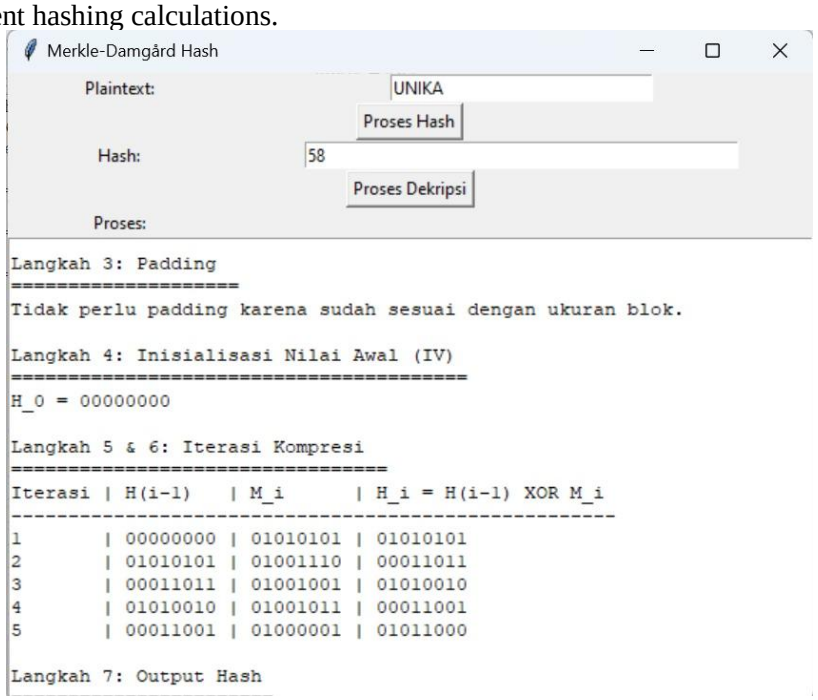

Figure 2. the process of hashing the text "UNIKA"

Displays the results of hashing iterations using XOR operations on each block. The final hashing result is displayed in binary and ASCII format as the final output.

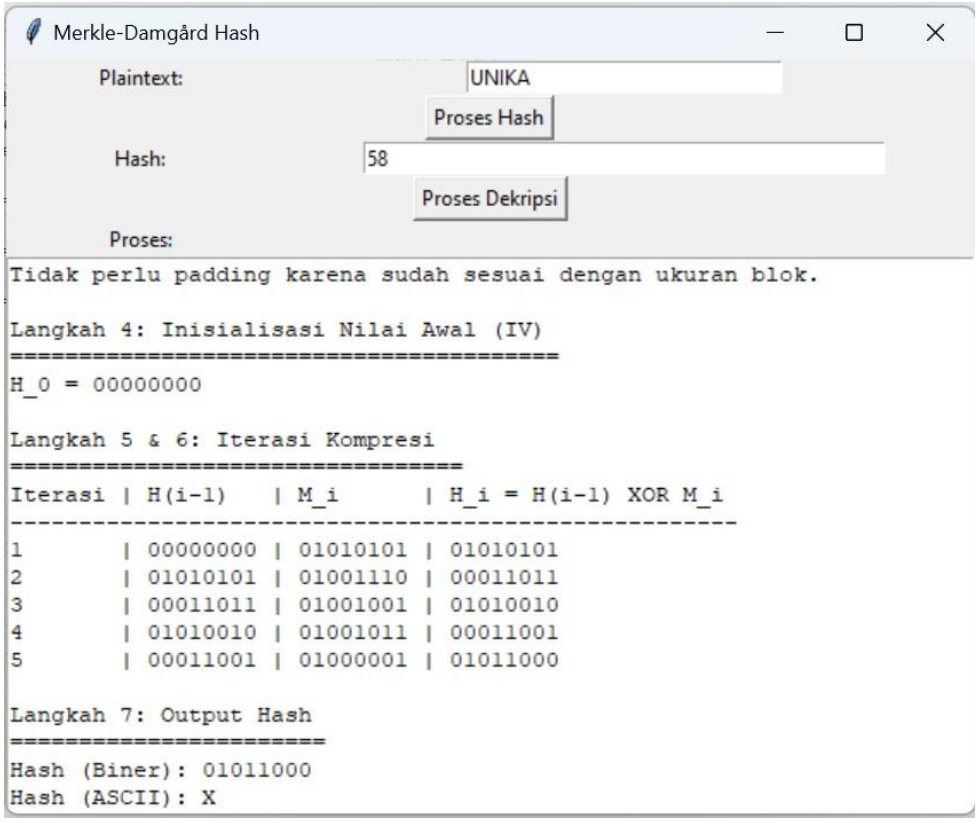

Figure 3.Hashing Results

The hashing process begins by converting each character in the text into a binary representation based on the ASCII code, then dividing it into 8-bit blocks. After that, the hashing process is carried out by initializing the initial value (IV) as 00000000 and applying the XOR operation iteratively to each binary block. Through a series of iterations, the hash value is updated by XORing the previous result and the next message block. The final result is a hash value in binary of 01011000, which when converted to ASCII becomes the character 'X'.

4. CONCLUSION

Merkle-Damgård Algorithmis one of the basic constructions in creating a cryptographic hash function. This algorithm works by dividing the message into small blocks and processing them iteratively using a compression function to produce the final hash value. The main advantage of this algorithm is its ability to maintain cryptographic security properties such ascollision resistanceAndpreimage resistance, provided that the compression function used has good security. The Merkle-Damgård algorithm is the foundation for many popular hash functions, such asMD5, SHA-1, and SHA-2**.**However, some Merkle-Damgård based algorithms have been shown to be weak against collision attacks (collision attack), so that more modern hash functions are likeSHA-3developed to improve security. The implementation of this algorithm inmanual calculations and testing using Pythonshows that its iterative process can produce unique and deterministic hash values, so it can be used in various data security applications.

REFERENCES

Ariesanda, B. (n.d.). *Analisis dan Pengembangan Merkle-Damgård Structure*.

Fajrin, A. M. (n.d.). *Perbandingan Performa Kecepatan dari Algoritma Hash Function untuk Proses Enkripsi Password* (Vol. 4, Issue 4).

Indrayani, R., Ferdiansyah, P., & Koprawi, M. (2024). *Analisis Penggunaan Kriptografi Metode AES 256 Bit pada Pengamanan File dengan Berbagai Format*. *4*(2). https://doi.org/10.47709/digitech.v4i2.5457

Patricia, J. (2024). *CYBER NOTARY DAN DIGITALISASI TANDA TANGAN*. Grup Penerbitan CV BUDI UTAMA.

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).

Tiwari, H. (2017). Merkle-Damgård Construction Method and Alternatives: A Review. In *Survey Paper JIOS* (Vol. 41, Issue 2).

Ariyus, D., & Wahidah, N. (2018). Analisis Perbandingan Keamanan Algoritma SHA-1 dan SHA-256 pada Aplikasi Keamanan Data. Jurnal Ilmiah Teknologi Informasi, 12(2), 45-56. (t.thn.).

Kurniawan, Y., & Setiawan, E. (2021). Analisis Keamanan Algoritma SHA-256 terhadap Serangan Tabrakan. Jurnal Ilmiah Teknik Informatika, 9(1), 34-42. (t.thn.).

Prasetyo, B., & Hidayat, R. (2019). Implementasi Algoritma MD5 untuk Integritas Data pada Sistem Keamanan Informasi. Jurnal Sistem Informasi dan Teknologi, 7(3), 123-130. (t.thn.).

Rahardjo, B., & Susanto, A. (2017). Penerapan Fungsi Hash dalam Pengamanan Data Transaksi Elektronik. Jurnal Informatika dan Komputer, 10(2), 89-97. (t.thn.).

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. (t.thn.).