Analysis of the Effect of Padding Schemes on Entropy, Bit Distribution, Hash Collisions, and Processing Time in Merkle-Damgård

Valois Vicenti Sirait[1], Mia Elisabet Malau[2], Jeni Percani Sinaga[3], Dian Pratama Gulo[4], Berkat Damai Halawa[5]
Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas, Indonesia

Article Info

Keywords:

Merkle Damgard, Padding, Hashing, Entropy, Hash Collision

ABSTRACT

Data security in cryptographic systems is highly dependent on the strength of the hashing algorithm. One of the most commonly used hashing structures is Merkle-Damgård, which converts the compression function into a fixed-size hashing function. The padding technique in this structure plays an important role in determining the bit distribution, entropy, and the probability of collision in the hash results. This study aims to analyze and compare three padding methods, namely 1 & 0 bit padding, repeating pattern padding (0xAA), and 1 bit padding (0xFF), based on bit distribution parameters, Shannon entropy, hash collision, and processing time. The results show that 1 bit padding (0xFF) has the highest Shannon entropy value (0.9940), indicating a better level of randomness compared to other methods. In terms of bit distribution, this padding also produces better balance than other paddings. However, the hash collision rate (74.90%) is still relatively high, indicating that the padding method alone is not enough to significantly reduce the probability of collision. In terms of time efficiency, padding bits 1 & 0 have the fastest execution time (0.000132 seconds), while padding bit 1 (0xFF) has the longest processing time (0.000177 seconds). With these results, it can be concluded that the padding method affects the hash characteristics, but does not significantly reduce the collision probability. Therefore, further optimization is needed to improve the security of Merkle-Damgård-based hashing.

Corresponding Author:

Valois Vicenti Sirait
Fakultas Ilmu Komputer, Universitas Katolik Santo Thomas, Indonesia
E-mail: valoissirait0@gmail.com

1. INTRODUCTION

Merkle-Damgård is one of the key constructions in cryptographic hash function design. Developed by Ralph Merkle and Ivan Damgård in 1989, this technique is used in various popular hash algorithms such as MD5, SHA-1, and SHA-2.(Ghanimi et al., 2024). One of the main components in this security system is, which functions to convert data into a unique representation with a fixed length.(Sitorus et al., 2024). The most common hashing structure used in modern algorithms is the Merkle-Damgård, a transformation that converts a compression function into a fixed-size hashing function.(Tiwari, 2017). Hashing algorithms such as MD5, SHA-1, and SHA-2 use this structure to ensure data security in a variety of applications, including authentication, digital signatures, and data integrity.(Anwar et al., 2021).

In the Merkle-Damgård based hashing process, padding technique plays a crucial role. Padding is a procedure used to adjust the input length to match the block size processed by the hashing algorithm. According to(Nandi, nd)that padding is very necessary for hashingMerkle Damgård. The padding method used can affect the bit distribution in the hash result, entropy, and the probability of collision. Therefore, the study of padding in the Merkle-Damgård structure is important because it can affect the effectiveness and security of the resulting hashing algorithm.

One of the main challenges in this research is to determine the optimal padding method to generate hashes with a high degree of randomness, minimize collisions, and ensure computational efficiency. Different padding techniques can produce different bit distributions, which ultimately affect the hash entropy and the probability of collisions. Therefore, a deeper understanding of the impact of different padding techniques on hashing results is essential to improve the security of systems using Merkle-Damgård structures.(Coron et al., nd).

Previous research has shown that padding schemes play an important role in determining entropy, bit distribution, security against hash collisions, and processing time efficiency in Merkle-Damgård-based algorithms. The development of secure and efficient padding schemes remains a major focus in modern cryptography research.(Ariesanda, nd).

This study aims to analyze and compare various padding methods used in Merkle-Damgård based hashing algorithms. The analysis is carried out based on several parameters, namely the distribution of bits in the hash results, the resulting entropy value, and the hash collision rate. By evaluating the effect of each padding method on these parameters, this study is expected to provide insight into the most effective padding method in improving the security and efficiency of hashing in various cryptographic applications. In addition, this study also aims to assess the processing efficiency of each padding method to ensure that higher security does not significantly sacrifice system performance.

Through this research, it is expected to gain a better understanding of how different padding methods work and how to choose the most secure and efficient padding in various cryptographic scenarios. Thus, the results of this study can contribute to the development of more secure, efficient, and resistant hashing algorithms against cryptographic attacks, as well as provide a foundation for further research in the field of information security.

2. RESEARCH METHODS

This study designs a research method that aims to analyze padding onMerkle-Damgård structure. Figure 1 shows the workflow of the research model conducted in this study, starting from data input to parameter evaluation.

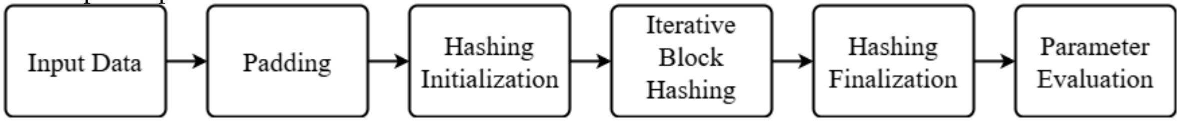

Figure 1. Research Flow Diagram

Data Input

Data input is the first stage in the hashing process or in other cryptographic algorithms, where the data to be processed is entered into the system. Input data can be text, numbers, images, or information that you want to protect or change into a more secure format, for example through a hash function.(Maysanjaya & Dermawan, 2024). For example, data such as passwords, messages, or documents to be hashed. In the Merkle-Damgård method, the input data is called a message (M) and then it will be broken into fixed-size blocks (b bits), which are then processed through a hash function to produce a secure hash value.

Padding

Padding is the process of adding additional data (usually bits or bytes) to input data to meet a certain size requirement by a hashing or cryptographic algorithm.(Harsa Kridalaksana & Arifin, 2016). Padding is important because many hashing algorithms require input in a specific block size (e.g., 512 bits or 1024 bits). Padding ensures that the length of the data fits into the required block. For example, a message(M) is split into blocks of a fixed size (b bits). If the length of the message(M) is not divisible by the block size, then padding must be done so that the total length of the data is a multiple of the block size. In the Merkle-Damgård scheme, there are several padding methods that can be used to ensure that the input length fits the block size required by the hash function. Padding plays an important role in determining how the data is padded before further processing. Some commonly used padding methods include:

1. Padding Bits 1 and 0

$$M^I = M \;||\; 1 \;||\; 0 \ldots 0 \;||\text{len}(M)$$

Explanation:
$M$is the original data to be hashed
|| is a concatenation operator that combines two pieces of data into one.
1(bit) describes padding starting by adding 1 bit at the end of the M data.
0..0 Indicates after bit 1, add zero bits according to multiples of the block size.
len(M) is the length of the original data in bits.

2. Padding With Repeating Pattern (0xAA)

$$M^I = M \;||\; 0\text{xAA}|| \ldots \;||\text{len}(M)$$

Explanation:
$M$is the original data to be hashed
||is a concatenation operator that combines two pieces of data into one.
0xAA is a hexadecimal number (10101010 in binary)
…adds repeating patterns (0xAA) in multiples of the block size
len(M) is the length of the original data in bits.

3. Padding with Bit 1 (Byte 0xFF)

$$M^I = M \;||\; 0\text{xFF}|| \ldots \;||\text{len}(M)$$

Explanation:
$M$is the original data to be hashed
||is a concatenation operator that combines two pieces of data into one.
0xFF is(11111111 in binary)
… adds repeating patterns (0xFF) in multiples of the block size
len(M) is the length of the original data in bits.

Hashing Initialization

Hashing Initialization is a stage that involves the initialization of the initial value of the hashing algorithm, which will be used to start the hash calculation process. Usually, this initial value is a predetermined constant value, which serves as a starting point for calculating the hash value. The hash function starts with an initial chaining value H0 which is usually determined by the algorithm (e.g., a constant value). H0 is referred to as the Initialization Vector (IV), which is usually initialized to a fixed value(Thampi et al., nd).

Iterative Block Hashing

Iterative block hashing is an iterative block hashing process that divides the input data into smaller blocks and then processes each block repeatedly (iteratively) to produce a hashing value. The hashing function will be applied to each block of data, and the result will be affected by the previous block, forming a more complex chain. Each Mi block is processed sequentially through the compression formula:

$$H_i = f(H_{i-1}, M_i)$$

Explanation:
H_i is the new hash value
$f$is a compression function that mixes 2 inputs and$H_{i-1}M_i$
$H_{i-1}$is the Chaining Value or hash value of the previous block.
M_i is the i-th data block that comes from the original message after being split into small blocks of fixed size.

Hashing Finalization

Hashing Finalization is the final stage in the hashing process, where the interim computation results are further processed to produce a final hash value ready for use.After all blocks are processed, the hash result is determined by the formula:

$$H_{Final} = H_1 \,||\, H_2 \,||\, H_3 \,||\ldots||\, H_n$$

Explanation:
$H_{Final}$is the hash result
$H_1,\ H_2\ ,\ H_3\ ,\ H_n$ is the hash value of each iteration (each padding block).
||is a concatenation operator that combines two pieces of data into one.

Parameter Evaluation

Parameter evaluation is the process of assessing the quality and security of a hash algorithm based on various technical aspects. This evaluation aims to ensure that the hash algorithm has characteristics that are safe, efficient, and resistant to cryptographic attacks. Some of the main parameters evaluated include Bit Distribution, Shannon Entropy, Hash Collision, Processing Time & Computational Cost(Waluyo, 2023).

a. Bit Distribution
Measures how random the bit distribution is in a hash result. A good hash should have an even bit distribution to prevent patterns that can be exploited by an attacker.
b. Shannon Entropy
Measures the degree of randomness and uncertainty in a hash result. The higher the entropy, the harder the hash is to predict or compress, making it more secure against statistical analysis attacks.
c. Hash Collision
When two inputs in a hash algorithm get the same hash value, it is called a "hash collision". Since hash functions have a fixed output length but accept inputs of infinite size, collisions always occur.
d. Processing Time
Processing time to determine the duration of time required to process and display the hash results of the message(M), providing an overview of the computational efficiency for each type of padding.

3. RESULTS AND DISCUSSION

The results and discussion were carried out using three paddings analyzed in this study, namely padding with bits 1 and 0, padding with a repeating pattern (0xAA) and padding with bit 1 (Byte 0xFF).

Results and Discussion Using Padding with bits 1 and 0

Data Input

In the data input, the message (M) and character block size are as follows:
M= GROUP TWO
Block size: 5 characters
M1=”KELOM”, M2=”POKDU”, M3=”A”

Padding

The original M3 message consists of only one character, so it needs to be padded to a block length of 5 characters (40 bits) with the following steps:

1. ASCII to binary conversion: "A" → ASCII(65) → 01000001
2. Add bit 1 after data: 01000001 1
3. Increase bit 0 to reach 40 bits: 01000001 10000000 00000000 00000000 00000000
4. Add message length: Length of "A" is 8 bits →binary: 00001000
5. Combine each binary, final result: 01000001 10000000 00000000 00000000 00001000
6. Convert the final result to decimal: [65, 128, 0, 0, 8]

Padding result M3 =[65, 128, 0, 0, 8]

Hashing Initialization
Initial chaining value (H0) = 0

Iterative Block Hashing
Block 1: M1=”KELOM”

$$H_1 = f(H_0, M_1 = (0 + \text{ASCII}('\text{K}') + \text{ASCII}('\text{E}') + \text{ASCII}('\text{L}') + \text{ASCII}('\text{O}') + \text{ASCII}('\text{M}'))\ mod\ 256$$

$$H_1 = (0 + 75 + 69 + 76 + 79 + 77)\ mod\ 256$$

$$H_1 = 376\ mod\ 256 = 120$$

Block 2: M2=”POKDU”

$$H_2 = f(H_1, M_2 = (0 + \text{ASCII}('\text{P}') + \text{ASCII}('\text{O}') + \text{ASCII}('\text{K}') + \text{ASCII}('\text{D}') + \text{ASCII}('\text{U}'))\ mod\ 256$$

$$H_2 = (120 + 80 + 79 + 75 + 68 + 85)\ mod\ 256$$

$$H_2 = 507\ mod\ 256 = 251$$

Block 3: M3= Padding Result [65,128,0,0,8]

$$H_3 = (251 + 65 + 128 + 0 + 0 + 8)\ mod\ 256$$

$$H_3 = 452\ mod\ 256 = 196$$

Hash Finalization

$H_{Final} = 120\ ||\ 251||\ 196$

The hash value of M="GROUPSECOND" is 120 251 196

Results and Discussion Using Padding With Repeating Patterns (0xAA)
Data Input

In the data input, the message (M) and character block size are as follows:
M= GROUP TWO
Block size: 5 characters
M1=”KELOM”, M2=”POKDU”, M3=”A”

Padding

The original M3 message consists of only one character, so it needs to be padded to a block length of 5 characters (40 bits) with the following steps:

1. ASCII to binary conversion: "A" → ASCII(65) → 01000001
2. Add pattern 0xAA (10101010) 40 bits: 01000001 10101010 10101010 10101010 10101010
3. Add message length: Length of "A" is 8 bits →binary: 00001000
4. Combine each binary, final result: 01000001 10101010 10101010 10101010 00001000
5. Convert the final result to decimal: [65, 170, 170, 170, 8]

Padding result M3 =[65, 170, 170, 170, 8]

Hashing Initialization
Initial chaining value (H0) = 0

Iterative Block Hashing
Block 1: M1=”KELOM”

$$H_1 = f(H_0, M_1 = (0 + \text{ASCII}('\text{K}') + \text{ASCII}('\text{E}') + \text{ASCII}('\text{L}') + \text{ASCII}('\text{O}') + \text{ASCII}('\text{M}'))\ mod\ 256$$

$$H_1 = (0 + 75 + 69 + 76 + 79 + 77)\ mod\ 256$$

$$H_1 = 376\ mod\ 256 = 120$$

Block 2: M2=”POKDU”

$$H_2 = f(H_1, M_2 = (0 + \text{ASCII}('\text{P}') + \text{ASCII}('\text{O}') + \text{ASCII}('\text{K}') + \text{ASCII}('\text{D}') + \text{ASCII}('\text{U}'))\ mod\ 256$$

$$H_2 = (120 + 80 + 79 + 75 + 68 + 85)\ mod\ 256$$

$H_2 = 507 \, mod \, 256 \; = \; 251$

Block 3: M3= Padding Result [65, 170, 170, 170, 8]

$H_3 = (251 + 65 + 170 + 170 + 170 + 8) \, mod \, 256$

$H_3 = \; 834 \, mod \, 256 \; = \; 66$

Hash Finalization

$H_{Final} = 120 \,||\, 251 || \, 66$

The hash value of M="GROUPSECOND" is 120 251 66

Results and Discussion Using Padding With Bit 1 (Byte 0xFF)

Data Input

In the data input, the message (M) and character block size are as follows:

M= GROUP TWO

Block size: 5 characters

M1=”KELOM”, M2=”POKDU”, M3=”A”

Padding

The original M3 message consists of only one character, so it needs to be padded to a block length of 5 characters (40 bits) with the following steps:

1. ASCII to binary conversion: "A" → ASCII(65) → 01000001
2. Add byte 0xFF (11111111) 40 bits: 01000001 11111111 11111111 11111111 11111111
3. Add message length: Length of "A" is 8 bits →binary: 00001000
4. Combine each binary, final result: 01000001 11111111 11111111 11111111 00001000
5. Convert the final result to decimal: [65, 255, 255, 255, 8]

Padding result M3 =[65, 255, 255, 255, 8]

Hashing Initialization

Initial chaining value (H0) = 0

Iterative Block Hashing

Block 1: M1=”KELOM”

$H_1 = f(H_0, M_1 = (0 + \text{ASCII}('\text{K}') + \text{ASCII}('\text{E}') + \text{ASCII}('\text{L}') + \text{ASCII}('\text{O}') + \text{ASCII}('\text{M}')) \, mod \, 256$

$H_1 = (0 + 75 + 69 + 76 + 79 + 77) \, mod \, 256$

$H_1 = 376 \, mod \, 256 \; = \; 120$

Block 2: M2=”POKDU”

$H_2 = f(H_1, M_2 = (0 + \text{ASCII}('\text{P}') + \text{ASCII}('\text{O}') + \text{ASCII}('\text{K}') + \text{ASCII}('\text{D}') + \text{ASCII}('\text{U}')) \, mod \, 256$

$H_2 = (120 + 80 + 79 + 75 + 68 + 85) \, mod \, 256$

$H_2 = 507 \, mod \, 256 \; = \; 251$

Block 3: M3= Padding Result [65, 255, 255, 255, 8]

$H_3 = (251 + 65 + 255 + 255 + 255 + 8) \, mod \, 256$

$H_3 = \; 1089 \, mod \, 256 \; = \; 65$

Hash Finalization

$H_{Final} = 120 \,||\, 251 || \, 66$

The hash value of M="GROUPSECOND" is 120 251 65

Parameter Evaluation

In the parameter evaluation process, the Python code implementation is carried out on Google Colab, because the message (M) used in this process must have a more significant length. Manual calculations only include simple examples to explain the basic concept of padding in the Merkle-

Damgård algorithm. However, to obtain more accurate and representative results of the influence of the analyzed padding on bit distribution, entropy, hash collision, and processing time, a longer message (M) is required. Therefore, testing is carried out using the Python code implementation, which is able to handle larger message sizes (M) and provide a more in-depth analysis of each evaluation parameter.

In the implementation of the Python code, in calculating Bit Distribution, the format() function is used to convert bytes into binary format and the count() function to count the number of bits 0 and 1 in a binary string. To calculate Shannon Entropy, the math.log2() function is used to calculate the base 2 logarithm and the count() function to count the frequency of occurrence of bits 0 and 1 in a binary string. In the Hash Collision test, the implemented hash function is used, namely (merkle_damgard_hash) to calculate the hash of the message, while the dictionary is used to track whether the resulting hash already exists, indicating a collision. Finally, to measure Processing Time, the time.time() function is used to record the time before and after the hash process, then calculate the difference to get the execution time which indicates the computational cost.

In the implementation of the Python code, the input data is specified as message(M) and character blocks as follows:

Message (M) = “RESEARCH JOURNAL OF CRYPTOGRAPHY AND STEGANOGRAPHY COURSE ON THE TOPIC OF PADDING ANALYSIS WITH ONE AND ZERO BITS, PADDING WITH REPEATING PATTERN AND PADDING WITH ONE BYTE IN MERKLE-DAMGARD: A COMPARATIVE STUDY OF THE BEST PADDING WITH EVALUATION PARAMETERS OF BIT DISTRIBUTION, SHANNON ENTROPY, HASH COLLISION AND PROCESSING TIME & . WRITTEN BY GROUP TWO: VALOIS VICENTI SIRAIT, MIA ELISABET MALAU, JENI PERCANI SITUMORANG, FIFTH SEMESTER STUDENTS, INFORMATICS ENGINEERING STUDY PROGRAM, FACULTY OF COMPUTER SCIENCE, SANTO THOMAS CATHOLIC UNIVERSITY, "SETIA BUDI STREET, TANJUNG SARI, MEDAN SELAYANG SUBDISTRICT, MEDAN CITY, NORTH SUMATERA, INDONESIA"

Character block = 64 characters

In the data hashing test, each type of padding analyzed is tested using several parameters to evaluate its performance. Here is a visualization of the parameter evaluation for each hash with the analyzed padding.

a. Bit Distribution

Bit distribution counts the number of bits 0 and 1 in the hash to assess the balance of the bit distribution. The bit distribution results show the difference in the balance between bits 0 and 1 in each type of padding. In padding with bits 1 & 0, there are 49 bits 0 and 39 bits 1. In the 0xAA pattern padding, there are 52 bits 0 and 36 bits 1. While in padding with bit 1 (0xFF), the bit distribution is more balanced, with 48 bits 0 and 40 bits 1. Analyzed, padding bit 1 (0xFF) produces a more balanced bit distribution, indicating that this padding produces a more random hash compared to other types of padding. Can be seen in Figure 2.

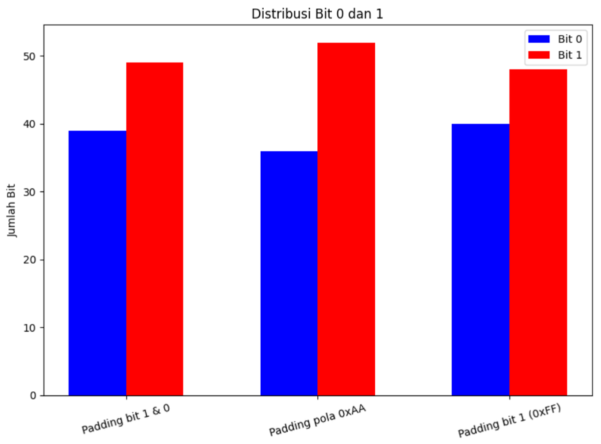

Figure 2. Bit Distribution Results

b. Shannon Entropy

Shannon entropy is used to measure the randomness of the distribution of bits in a hash value, where the higher the entropy value, the more random the result.(Supriyatna, nd). The results of the Shannon entropy measurement show differences in the level of randomness between each type of padding. In padding with bits 1 & 0, the entropy value was recorded at 0.9907, while in the 0xAA pattern padding, the entropy was slightly lower, which was 0.9760. Padding with bit 1 (0xFF) produced the highest entropy value, which was 0.9940. Analyzed, padding bit 1 (0xFF) has the highest entropy value, indicating that the resulting hash is more random and more difficult to predict, providing better security in the context of data processing. Can be seen in Figure 3.

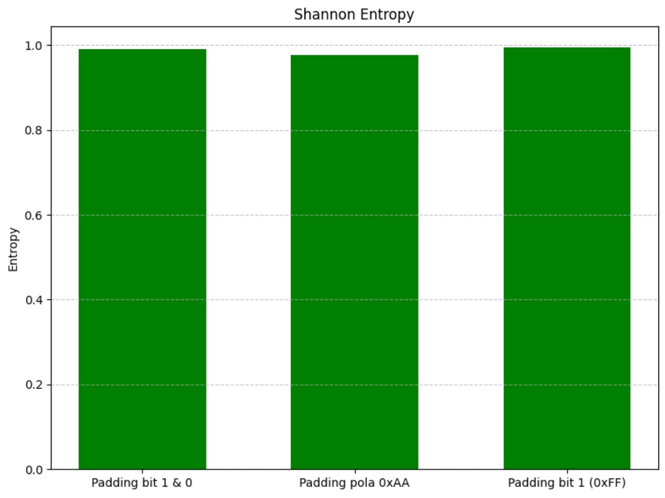

Figure 3. Bit Distribution Results

c. Hash Collision

Hash Collision is tested by generating multiple hashes for different messages and checking whether there are any identical hash values, indicating potential security issues. The collision rate test results show that padding bit 1 (0xFF) has a slightly lower collision rate (74.90%) compared to padding bits 1 & 0 (75.30%) and padding pattern 0xAA (75.00%). However, the high collision rate for all three types of padding indicates that the hash function used is still susceptible to possible collisions. This indicates that improvements are needed in the design of the hash function to reduce the possibility of collisions and improve overall security. This can be seen in Figure 4.

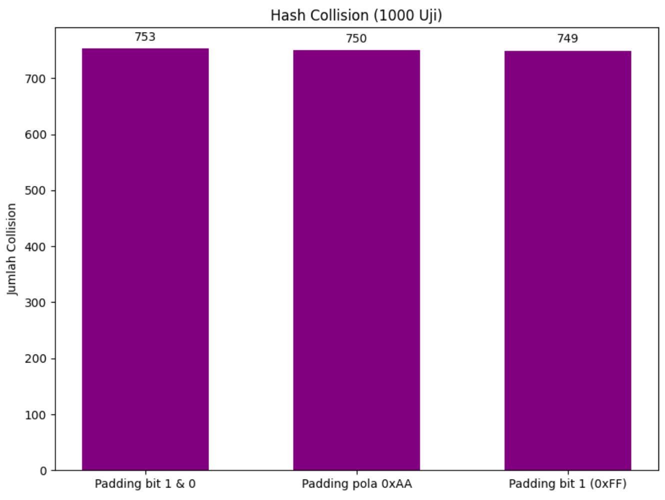

Figure 4. Hash Collision Results

d. Processing Time

Processing time to determine the duration of time required to process and display the hash results of the message (M), provides an overview of the computational efficiency for each type of padding. The evaluation results show that padding bits 1 & 0 have the fastest processing time, which is 0.000132 seconds, followed by padding pattern 0xAA with a time of 0.000157 seconds, and padding bit 1 (0xFF) with a time of 0.000177 seconds. Can be seen in Figure 5.

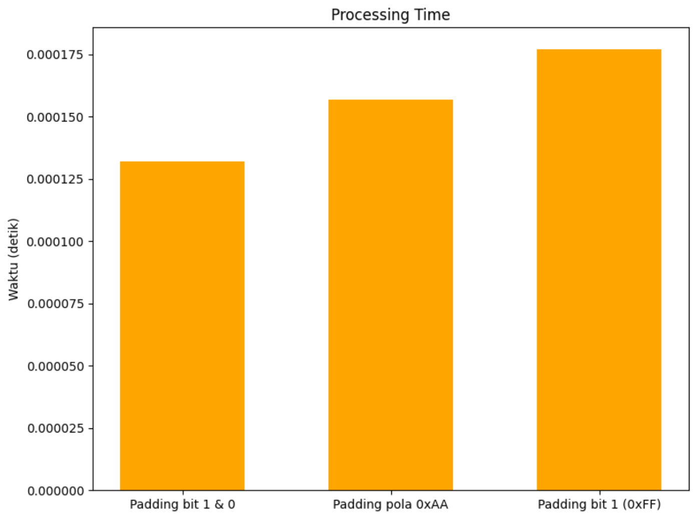

Figure 5. Hash Collision Results

A summary of the research results on padding analysis in Merkle-Damgård is shown in Table 1. This table presents a comparison of the total bits, bit distribution, Shannon entropy value, the number of hash collisions in the test, the collision rate (%), and the processing time for each type of padding. From this data, it can be analyzed how each type of padding affects the characteristics of the resulting hash, including the balance of bit distribution, the degree of randomness, the probability of collision, and the efficiency of execution time.

Table 1. Parameter Evaluation Results

Parameter	**Padding bits 1&0**	**Padding pattern 0xAA**	**Padding bit 1 (0xFF)**
Total Bit	88	88	88
Bit Distribution (0/1)	49 / 39	52 / 36	48 / 40
Shannon Entropy	0.9907	0.9760	0.9940
Hash Collision (tests)	753	750	749
Collision Rate	75.30%	75.00%	74.90%
Processing Time (sec)	0.000132	0.000157	0.000177

4. CONCLUSION

Based on the research results obtained, it can be concluded that the type of padding used has an effect on the characteristics of the resulting hash. In terms of bit distribution, padding bit 1 (0xFF) has a more balanced bit distribution (48 bits 0 and 40 bits 1), which indicates a better level of randomness compared to other paddings. Shannon entropy also shows that padding bit 1 (0xFF) has the highest entropy value (0.9940), indicating a more unpredictable hash. However, in the hash collision test, padding bit 1 (0xFF) is only slightly better with a collision rate of 74.90%, compared to padding bits 1 & 0 (75.30%) and the 0xAA pattern (75.00%). However, this collision rate is still relatively high, indicating that the hash function can be further improved to reduce the likelihood of collisions. In terms of processing time, padding bits 1 & 0 have the fastest execution time (0.000132 seconds), followed by padding pattern 0xAA (0.000157 seconds) and padding bit 1 (0xFF) (0.000177 seconds). The difference in processing time is very small and insignificant, so time efficiency is not the main factor in determining the best padding. Overall, padding bit 1 (0xFF) gives better results in terms of bit distribution balance and entropy, although the difference is not too far from other paddings. However, the high collision rate indicates that further optimization of the hash function is still needed to improve its security and uniqueness.

REFERENCE

Anwar, M. R., Apriani, D., & Adianita, I. R. (2021). Hash Algorithm In Verification Of Certificate Data Integrity And Security. *APTISI Transactions on Technopreneurship*, *3*(2), 67–74. https://doi.org/10.34306/att.v3i2.212

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

Coron, J.-S., Dodis, Y., Malinaud, C., & Puniya, P. (n.d.). *Merkle-Damgård Revisited : how to Construct a Hash Function*.

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

Harsa Kridalaksana, A., & Arifin, Z. (2016). 8. SaKTI Henry_KRIPTOGRAFI AES MODE CBC PADA CITRA DIGITAL BERBASIS ANDROID. In *Prosiding Seminar Ilmu Komputer dan Teknologi Informasi* (Vol. 1, Issue 1).

Maysanjaya, I. M. D., & Dermawan, K. T. (2024). *MANAJEMEN BASIS DATA (Teori dan Implementasi)* (I). PT. Sonpedia Publishing Indonesia.

Nandi, M. (n.d.). *Characterizing Padding Rules of MD Hash Functions Preserving Collision Security*.

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).

Supriyatna, A. (n.d.). Analisis Bibliometrik Shannon Entropy: Tren Penelitian dan Relevansi Multidimensional. In *Jurnal Infortech* (Vol. 6, Issue 2). http://ejournal.bsi.ac.id/ejurnal/index.php/infortech

Thampi, S. M., Gelenbe, E., Atiquzzaman, M., Chaudhary, V., & Li Editors, K.-C. (n.d.). *Lecture Notes in Electrical Engineering 735* (Vol. 1). http://www.springer.com/series/7818

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

Waluyo, T. (2023). *Manajemen Agrobisnis di Era Society 5.0* (K. Retnawati, Ed.; I). CV. Mega Press Nusantar.