Analyzing Hash Collision Probability in Password Storage using the Birthday Paradox

Daniel Putra Rywandi S - 13524143
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: Danielputrarywandisa@gmail.com , 13524143@std.stei.itb.ac.id

***Abstract*— As modern systems increasingly rely on digital authentication, passwords continue to serve as a primary method for securing access and user identities. To mitigate the risk of data breaches, computer systems do not store passwords in plain text, but rather as the output of cryptographic hash functions, which are designed to be one-way and computationally irreversible. Hash functions are constructed to be computationally infeasible to reverse and resistant to collisions—situations where two different inputs produce the same hash output. However, according to the Birthday Paradox in probability theory, such collisions can occur much earlier than commonly expected, especially as the number of hashed inputs increases.**

Keywords—combinatorics, hash function, password, birthday paradox, probability

I. Introduction

A hash function is an algorithm that takes an input message of arbitrary length and produces a fixed-length output, known as the hash value or digest, which uniquely represents the original input. Ideally, a secure hash function should satisfy the following four requirements:

1. It should be infeasible to generate a specific digest from an input.
2. It should be impossible to reconstruct the original message from its hash.
3. Slight changes in the input should produce drastically different outputs.
4. The resulting digest must have a fixed length, regardless of the input size.

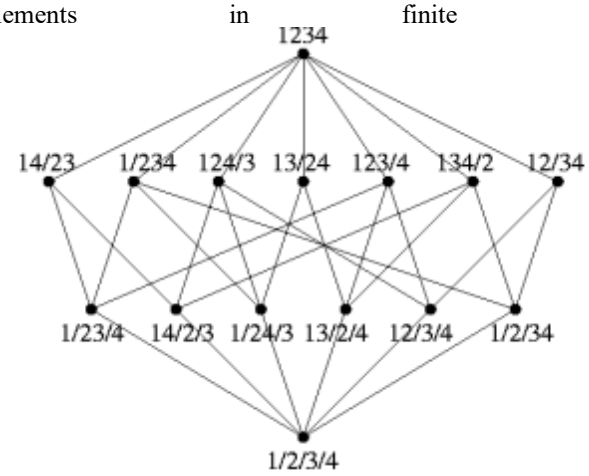
In practice, hash functions are expected to be collision-resistant, meaning that it should be computationally infeasible to find two different inputs that produce the same hash value. However, this resistance has a probabilistic limitation. The likelihood of collisions can be analyzed using the Birthday Paradox, a principle in probability theory. The paradox states that in a group of just 23 people, the probability of at least two sharing the same birthday exceeds 50%. Although there are 365 possible birthdays, the number of possible unique pairs grows quadratically with the number of individuals, making a match surprisingly likely.

Among the most commonly used cryptographic hash functions are MD5, SHA-1, and SHA-256, each differing in output size and level of security.

II. Theory

A. Combinatorics

Combinatorics is a branch of discrete mathematics concerned with counting, arrangement, and selection of elements in finite sets.



Gambar 1: Combinatorics,
(sumber : [1])

One of its fundamental concepts is the binomial coefficient, which expresses the number of ways to choose k elements from a set of n elements, disregarding the order of selection. It is defined as:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

In the context of hash collisions, combinatorics allows us to estimate how many unique pairs can be formed from a group of hashed inputs. Specifically, the number of possible pairs among n items is given by $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ which counts the number of ways to choose two distinct inputs without considering their order. The factor of 2 in the denominator eliminates duplicate pairings. This quadratic growth in the number of input pairs means that the probability of a collision increases rapidly as

more inputs are introduced, even when the number of possible hash outputs is extremely large.

B. The Pigeonhole Principle

The Pigeonhole Principle states that if more items are distributed into fewer containers, then at least one container must contain more than one item. Formally, if n items are placed into m containers and $n > m$, then at least one container must hold at least two items.

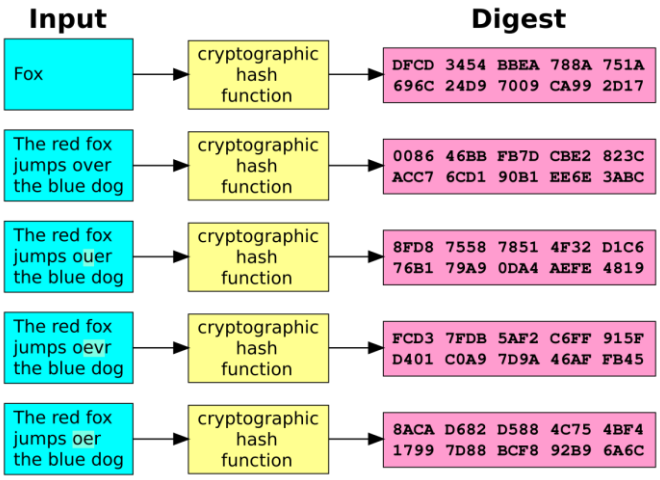
Gambar 2: The Pigeon princeple,
(sumber : [2])

In password hashing, the set of possible inputs such as all potential passwords—is virtually infinite, while the set of possible hash outputs is finite and fixed in size. For example, 2^{128} for MD5 or 2^{256} for SHA-256). As a result, the Pigeonhole Principle guarantees that some distinct inputs must produce the same output hash, known as a collision.

In practice, this principle forms the theoretical foundation for understanding why perfect collision resistance in hash functions is impossible. Regardless of how strong the hash algorithm is, as long as the number of possible inputs exceeds the number of unique outputs, collisions are not just possible—they are inevitable. Therefore, the Pigeonhole Principle is critical for justifying the need for strong hash designs that delay or make collisions computationally infeasible to find, even though they are mathematically guaranteed to exist.

C. Cryptographic Hash Functions

A hash function is an algorithm that processes an input of arbitrary length and converts it into a fixed-length output known as a hash value or digest. In security applications, cryptographic hash functions are used. They are designed to be one-way functions, meaning it is computationally infeasible to reconstruct the original input from its hash. The evolution of these functions has been driven by an ongoing effort to resist increasingly sophisticated computational attacks. Among the most historically significant are MD5, SHA-1, and SHA-256.



Gambar 3: The Pigeon princeple,
(sumber :
https://en.wikipedia.org/wiki/Cryptographic_hash_function)

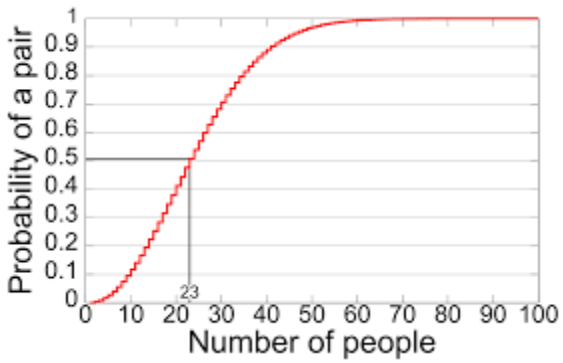
- Message Digest 5 (MD5) was designed by Ronald Rivest in 1991 and published in 1992 as a replacement for the earlier MD4 algorithm. It processes input data in 512-bit blocks and produces a 128-bit hash value. The core of the algorithm involves processing each block through four rounds of complex transformations. These rounds use non-linear logical functions, modular additions, and bitwise left shifts to thoroughly mix the input data. Despite its widespread initial adoption for data integrity verification, MD5 is now considered cryptographically broken and wholly insecure for any security-sensitive application. By 2004, researchers had demonstrated practical techniques for finding collisions, where two different inputs could be deliberately crafted to produce the same 128-bit hash. Because of this demonstrated susceptibility to collision attacks, MD5 is no longer recommended for uses such as digital signatures or password storage.
- Secure Hash Algorithm 1 (SHA-1) was developed by the United States National Security Agency (NSA) and published by NIST in 1995, SHA-1 was designed to improve upon the security of MD5. Like MD5, it processes 512-bit input blocks, but it produces a longer 160-bit hash output. Its internal structure is more complex, involving 80 rounds of processing that apply logical functions, constant additions, and bitwise operations to the data. The longer hash length was intended to offer greater resistance to brute-force and collision attacks. However, over time, theoretical weaknesses in SHA-1 were discovered, and it is now also considered insecure. The pivotal moment came in 2017 when researchers from CWI Amsterdam and Google announced the first practical collision attack, codenamed "SHAttered". They successfully created two different PDF files that produced the exact same SHA-1 hash, proving that the algorithm could no longer be trusted for applications requiring collision resistance. As a result, NIST has officially deprecated

SHA-1, and it is no longer recommended for use in secure cryptographic applications.

- Secure Hash Algorithm 256 (SHA-256) also designed by the NSA and published by NIST in 2001. It operates on 512-bit input blocks and generates a robust 256-bit hash value, offering a significantly larger output space to resist attacks. The algorithm processes each block through 64 rounds of operations, including bitwise rotations, logical functions, and modular additions with predefined constants. It is built upon the Merkle–Damgård construction, a design principle that allows it to securely process variable-length inputs. An essential property of its internal structure is a strong avalanche effect, where even a single-bit change in the input leads to a drastically different hash output. Currently, SHA-256 offers strong collision resistance and is regarded as a secure and reliable standard. While theoretical collisions must exist due to the finite output size, no practical collision attacks have been discovered to date. Consequently, SHA-256 is widely used in many critical security applications, including TLS for web security, digital signatures, and securing transactions in blockchain systems like Bitcoin.

D. The Birthday Paradox

The Birthday Paradox is a concept in probability theory that explains how surprisingly few elements are required in a set before a collision becomes likely. The classic version of the paradox states that in a group of just 23 people, there is over a 50% chance that two people share the same birthday, even though there are 365 possible birthdays. This paradox illustrates that the probability of a shared value increases with the number of elements compared to the total number of possible outcomes a principle directly applicable to cryptographic hash functions.



Gambar 4: The Birthday paradox,
(sumber : https://en.wikipedia.org/wiki/Birthday_problem)

Mathematically, the probability that no two people share the same birthday in a group of n people is approximately

$$P(no\ collision) \approx e^{-n^2/2(k)}$$

Where k is the total number of possible outcomes. As n increases, this probability rapidly decreases, and the probability of at least one collision increases accordingly.

This phenomenon has a direct parallel in the analysis of cryptographic hash functions. In the context of hashing, the “birthdays” are hash values, and each "person" represents a distinct input being hashed. Even though a hash function like SHA-256 has 2^{256} possible outputs, the Birthday Paradox implies that after hashing about 2^{128} inputs, there is a 50% chance of at least one collision occurring. This is known as the birthday bound, and it sets a theoretical limit on the collision resistance of hash functions.

E. The Birthday Attack

The Birthday Attack is a cryptographic technique that takes advantage of the counter-intuitive probability theory known as the Birthday Paradox. Rather than attempting to reverse a hash (as in a pre-image attack), the goal of a birthday attack is to find two different inputs that produce the same hash output, resulting in what is known as a collision.

This type of attack exploits the mathematical fact that collisions become significantly more likely as the number of inputs increases, even if the output space of the hash function is very large. The logic behind this is similar to the birthday paradox, which shows that in a group of just 23 people, there is a greater than 50% chance that at least two individuals share the same birthday. Likewise, with hash functions, it is much easier to find any two inputs that hash to the same value than to find a specific input that matches a given hash.

In practice, a birthday attack involves generating a large number of inputs, computing their hashes, and comparing the outputs to detect any matches. This process is often automated using hash tables or dictionaries to efficiently store and check for collisions. While such attacks can be computationally expensive, they are feasible against hash functions with insufficient output size or known weaknesses.

III. Implementation

A. Objective

The purpose of this implementation is to compare the behavior of two well-known cryptographic hash algorithms, MD5 and SHA-1, by simulating and detecting hash collisions using a birthday paradox–based approach. The simulation works by generating random input strings and checking if any two of them produce the same hash output.

To make collisions practically observable within a reasonable amount of time and computational resources, the simulation applies hash truncation, where only the first few bits of the hash output (e.g., 16 or 24 bits) are considered. This effectively reduces the output space and increases the probability of collisions, making the results more measurable in small-scale experiments. Although full-length hashes like MD5 and SHA-1 offer extremely large output spaces, such high entropy makes it computationally infeasible to detect natural collisions without excessive input trials.

B. Simulation

The simulation was developed using the Python programming language, utilizing the built-in hashlib library to

perform cryptographic hash operations. The core objective of the implementation is to simulate a collision scenario for both MD5 and SHA-1 algorithms by hashing randomly generated input strings and checking for repeated outputs.

To make the experiment computationally feasible, especially considering the enormous size of the full output space of MD5 (2^{128}) and SHA-1 (2^{160}), the simulation uses hash truncation. This means that only the first few hexadecimal characters of each hash output are compared. Truncation reduces the effective hash space from billions of possibilities to a few thousand, which significantly increases the chance of collisions within a limited number of attempts, making the experiment both observable and practical.

```python
import hashlib
import random
import string
import time

#config
STRING_LENGTH = 16
MAX_ATTEMPTS = 100_000_000
TRUNCATE_HEX = 12

def generate_random_string(length=STRING_LENGTH):
    characters = string.ascii_letters + string.digits
    return ''.join(random.choices(characters, k=length))

def get_truncated_hash(hash_func, input_str, hex_length=TRUNCATE_HEX):
    full_digest = hash_func(input_str.encode()).hexdigest()
    return full_digest[:hex_length]

def simulate_collision(hash_func, hash_name, truncate_to=TRUNCATE_HEX, max_attempts=MAX_ATTEMPTS):
    seen_digests = set()
    start_time = time.time()

    for attempt in range(1, max_attempts + 1):
        random_input = generate_random_string()
        digest = get_truncated_hash(hash_func, random_input, truncate_to)

        if digest in seen_digests:
            elapsed = time.time() - start_time
            print(f"[{hash_name}] Collision found after {attempt:,} attempts in {elapsed:.2f} seconds.")
            print(f"Truncated Digest: {digest}")
            print(f"Input Length: {STRING_LENGTH} characters")
            return attempt, elapsed, digest

        seen_digests.add(digest)

    elapsed = time.time() - start_time
    print(f"[{hash_name}] No collision found after {max_attempts:,} attempts. Time: {elapsed:.2f} seconds.")
    return None, elapsed, None
```

```python
def run_simulation():
    print("=== Hash Collision Simulation ===")
    print(f"Truncating hash output to {TRUNCATE_HEX * 4} bits ({TRUNCATE_HEX} hexadecimal characters)\n")

    md5_result = simulate_collision(hashlib.md5, "MD5")
    print()
    sha1_result = simulate_collision(hashlib.sha1, "SHA-1")
    print()
    print("=== Summary ===")
    if md5_result[0]:
        print(f"MD5: Collision at {md5_result[0]:,} attempts, Time: {md5_result[1]:.2f}s, Digest: {md5_result[2]}")
    if sha1_result[0]:
        print(f"SHA-1: Collision at {sha1_result[0]:,} attempts, Time: {sha1_result[1]:.2f}s, Digest: {sha1_result[2]}")

if __name__ == "__main__":
    run_simulation()
```

C. *Result*

To evaluate the effectiveness of the hash collision simulation, the program was executed separately for both MD5 and SHA-1 using a truncated 48-bit hash output. For each execution, random alphanumeric strings were continuously generated and hashed until a duplicate truncated hash value was found, indicating a collision.

In the case of MD5, a collision was observed after approximately 25,471,795 attempts, which is consistent with the birthday bound prediction for a 48-bit output. The time required to reach this collision was approximately 52.82 seconds. This demonstrates how reducing the hash space significantly increases the likelihood of collisions, even when using cryptographic hash functions that are otherwise secure under full output conditions.

On the other hand, testing with SHA-1 also produced a collision after approximately 30,867,722 attempts, with a total execution time of 62.72 seconds. While SHA-1 offers a larger native output size of 160 bits, truncating its output to 48 bits made it equally susceptible to birthday collisions, as expected from theoretical calculations.

Hash	Output Length	Collision Found	Attempts Until Collision	Time	Digest
MD5	48 bits	Yes	25,471,795	52.82 S	704ae0b3f2c2
SHA-1	48 bits	Yes	30,867,722	62.72 S	77ee68ccc5d1

These results clearly demonstrate the mathematical principle behind the Birthday Paradox, where collisions become increasingly likely as more inputs are hashed, and especially when the size of the hash output is reduced.

D. *Analysis*

The results obtained from the simulation align closely with the theoretical expectations described in the Birthday Paradox. When the hash output is truncated to 48 bits, the birthday bound suggests that a collision is likely to occur after approximately:

$$n \approx 1.17 \cdot \sqrt{2^{48}} \approx 19.4 \, million \, inputs$$

While slightly above the theoretical bound, these values are within a reasonable margin due to the randomness involved in the simulation. The results illustrate how the number of unique input pairs grows quadratically with each additional input, following the formula:

$$\frac{n(n-1)}{2}$$

This aligns with the pigeonhole principle, which ensures that collisions must eventually occur when more items (inputs) are mapped into fewer containers (hash values).

Ultimately, the implications of this study highlight a vital truth: security in digital systems is not only a function of algorithm design, but also of mathematical understanding. Even though hash functions are designed to be collision-resistant, the finite nature of their output space means collisions are not only possible they are inevitable given enough inputs. This is guaranteed by the pigeonhole principle, which states that if more items are mapped into fewer containers, at least one container will hold more than one item. As demonstrated by the birthday paradox, the number of input pairs grows quadratically, meaning that even with a seemingly large output space, such as 2^{48} for truncated hashes, the probability of collision increases rapidly. In practice, while a full 128-bit or 160-bit hash space might make collisions computationally infeasible, truncating the output dramatically reduces this security margin. The simulation results, where collisions occurred after around 25 to 30 million inputs, closely follow the theoretical birthday bound of approximately 19 million for a 48-bit space, confirming the mathematical predictions. This analysis reinforces the

importance of discrete mathematical reasoning in understanding and evaluating the real-world reliability of cryptographic functions.

This further emphasizes that cryptographic design must not rely solely on algorithmic complexity, but also account for statistical behavior over large-scale input usage. In modern applications such as password storage, blockchain, and digital signatures, the number of hashed values can grow rapidly over time. Without considering the mathematical inevitability of collisions, systems may appear secure while being statistically vulnerable. By integrating discrete mathematics into both theoretical and empirical analysis, this study not only validates the predictions of combinatorics and probability theory, but also demonstrates their practical consequences. As data volume continues to increase in today's digital landscape, understanding these mathematical foundations becomes not just beneficial—but essential—for building secure and reliable cryptographic systems.

IV. Conclusion

In order to understand how hash collisions occur and why their probability increases as more inputs are added, it is essential to first examine the mathematical foundations that govern such behavior. This study explored the discrete mathematical principles underpinning collision probability, particularly through the lens of combinatorics, cryptographic hash functions, and the birthday paradox. These theories provide a structured way to estimate, model, and explain how seemingly secure systems can yield unexpected outcomes due to the inherent limits of finite hash spaces.

Even though cryptographic hash functions are meticulously designed to resist collisions, mathematics reveals that such resistance has practical and probabilistic limits. In theory, a perfectly collision free hash function would require an infinite output space something unattainable in real world computing. Because every hash function maps a large, potentially infinite set of inputs to a finite set of outputs, the pigeonhole principle guarantees that collisions are not just possible—they are inevitable when enough inputs are processed.

This fact is often unintuitive to system designers, who may assume that a large bit length (such as 128-bit or 160-bit outputs) ensures complete uniqueness. However, as demonstrated through the birthday paradox and confirmed by empirical simulation in this paper, the number of required inputs to achieve a high probability of collision is far lower than the total number of possible hash outputs. This gap between theoretical capacity and practical collision likelihood is at the core of why probabilistic analysis is essential in cryptography.

Ultimately, the implications of this study highlight a vital truth: security in digital systems is not only a function of algorithm design, but also of mathematical understanding. By embracing the predictive power of discrete mathematics, system architects can more accurately assess the limitations of hashing mechanisms, and ensure that safeguards such as salting, longer hash outputs, and modern algorithms (e.g., SHA-256 or SHA-3) are employed appropriately to minimize real-world vulnerabilities.

V. Acknowledgment

With sincere gratitude, the author would like to thank God Almighty, whose blessings and guidance enabled the successful completion of this paper entitled “Analyzing Hash Collision Probability in Password Storage using the Birthday Paradox.”

The author extends heartfelt appreciation to Dr. Ir. Rinaldi, M.T., as the lecturer of IF2120 Discrete Mathematics. His teaching, insights, and the learning materials he has shared publicly have greatly supported the author's understanding of the theoretical concepts explored in this paper.

The author would also like to express sincere thanks to family and friends for their constant support, encouragement, and motivation throughout the writing process. Their presence has provided invaluable emotional strength during the completion of this academic task.

Finally, the author wishes to thank all individuals—whether mentioned or not—who have contributed directly or indirectly to the completion of this work.

References

[1] Rinaldi Munir, “Kombinatorika (Bagian 1)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian1-2024.pdf, Accessed Jun. 18, 2025, at 20:00.

[2] Rinaldi Munir, “Kombinatorika (Bagian 2)”, https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/19-Kombinatorika-Bagian2-2024.pdf, Accessed Jun. 18, 2025, at 22:00.

[3] S. Mehta, “Birthday attack in cryptography,” GeeksforGeeks,[Online], https://www.geeksforgeeks.org/ethical-hacking/birthday-attack-in-cryptography/, Accessed Jun. 18, 2025, at 22:00.

[4] Computerphile, “Birthday Paradox - Numberphile,” YouTube, May 2, 2013. [Online]. https://www.youtube.com/watch?v=jsraR-el8_o, Accessed Jun. 19, 2026, at 19:00

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Daniel Putra Rywandi S 13524143