The Gap Between Theoretical and Effective Password Security Using Combinatorial Analysis of Human Password Selection Patterns

Denzel Santoso - 13525014
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: denzelsan07@gmail.com , 13525014@std.stei.itb.ac.id

***Abstract*— As passwords remain a primary defense mechanism for securing personal information and data, their strength is critical for individual privacy. However, to mitigate exploitation and hacking, users are increasingly forced to create longer and more complex passwords. Due to the explosion of user accounts and the increasing complexity of password rules, users struggle to find ways to make up sufficiently secure yet easy-to-remember credentials. This burden frequently leads to users using predictable character patterns, dictionary words, and structural repetitions. This paper aims to evaluate the impact of such human behaviors on overall password security using combinatorial analysis. By applying fundamental counting principles, permutations, combinations, the Principle of Inclusion-Exclusion, and the Pigeonhole Principle, this paper quantifies the specific vulnerabilities introduced by human psychology. Using combinatorics to prove that satisfying a length requirement via repetition drastically reduces the combinatorial search space. Furthermore, a Python-based simulation is provided to quantify the mathematical gap between theoretical security maximums and human-generated passwords.**

Keywords—Password Security; Combinatorial Analysis; User Behaviour; Repetitive Patterns, Python Simulation

I. Introduction

From a strictly computational perspective, generating a secure password is a straightforward exercise in maximizing string length and alphabet size to create a large search space. However, authentication systems are not operated by perfect random number generators, they are used by humans.

Fig 1.1 Password Illustration [5]

(Taken from https://online.champlain.edu/blog/when-password-good)

Everyone has their password patterns. It might be their name, the name of the pet, a birthday, or a favorite word and number at the end all which has the same pattern reused across several accounts because their easy to remember but not a password that is hard to guess. These choices may feel practical and harmless in daily life, but they are what makes password vulnerable and easier for hackers to predict. A study from ScienceDirect found that user rely on common passwords, login name, and other meaningful combinations, while research evaluating password behaviors found that people frequently add numbers at the end of passwords and capital letters at the beginning [4].

As administrators enforce stricter length requirements, users adapt by repeating substrings. Recent large-scale analyses of leaked datasets prove that users frequently employ both short and long repetitive patterns to artificially inflate password length without increasing cognitive load.

Password security relies entirely on complexity—specifically, the number of possibilities an attacker must compute. In a comprehensive study of anonymized passwords, Bonneau estimated that human-generated passwords provide fewer than 10 bits of security against online guessing attacks and about 20 bits against offline dictionary attacks [1]. This is exponentially weaker than the theoretical strength implied by their length.

Combinatorial analysis provides the mathematical framework necessary to calculate entropy and key space. This paper applies discrete mathematics to demonstrate the theoretical boundaries of password security and explicitly quantify how human behavior subverts them.

II. Theoritical Foundation

Combinatorics is the branch of discrete mathematics that is used to count the amount of arrangement without trying every single possible arrangement. In cybersecurity, it serves as the mathematical engine for calculating entropy and theoretical resistance against brute-force attacks [2].

A. Fundamental Counting Principle

The foundational rule is the Rule of Product combined with the Rule of Sum for calculating password possibilities.

- **The Rule of Product**

  If an experiment has p outcome and the next is q outcome. The combined experiments have $p \times q$ outcomes, here we can understand that "and" has the same meaning of multiplying both outcomes.

  In the context of passwords, if a password has a total of L characters and each character is chosen from alphabet with the size of N, the total possible combinations is:

  $$Total\ Possibilities = N^L$$

  Example if a password has 8 characters (length) and 10 size for each character, then the total possibilities:

  $$Total\ Possibilities = 10^8 = 100.000.000$$

- **The Rule of Sum**

  If an outcome can be achieved by either p results from one method or q results from another, the total outcomes are $p + q$. In combinatorics, the word "or" represents addition.

  Example when you want to buy a new laptop, and the store has 5 models of Windows laptop and 3 models of Mac laptops. Because you can only choose either a Windows or a Mac, your total options is 8.

  $$Total\ Options = 5 + 3 = 8$$

- **Combine Both Rules**

  Real-world problems usually require both rules together.

  Example you are choosing an outfit. You have 3 shirts, 4 pairs of pants, and 2 hats. However, you decided either to wear a hat or not.

  1. Shirts and Pants : $3 \times 4 = 12$
  2. With a hat : $12 \times 2 = 24$
  3. Without a hat : $12 \times 1 = 12$
  4. Total Options (using Rule of Sum) : $12 + 24 + 12 = 48$

B. Permutations and Combinations

If a password can't use repeated characters, permutation and combinations is applied.

- **Permutation**

  Focus on ordered arrangement where the number of arrangements of r elements selected from n elements is denoted as $P(n, r)$.

  $$P(n, r) = \frac{n!}{(n-r)!}$$

- **Combination**

  Unordered arrangement where r elements from n elements is denoted as $C(n, r)$.

  $$C(n, r) = \frac{n!}{r!\,(n-r)!}$$

C. The Principle of Inclusion-Exclusion

Many systems required users to make a password that contain at least one number or at least one specific character. To calculate the exact space of valid passwords the Principle of Inclusion-Exclusion is needed. For two sets of A and B, the size of the union is:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

For three sets A, B, and C, the principle expands to :

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

D. The Pigeonhole Principle

The Pigeonhole Principle states that if you have more items (pigeons) than the available slot (pigeonhole), at least one slot must contain more than one items. The Generalized Pigeonhole Principle states that if M objects are placed into n boxes, at least one box must contain a minimum of $\lceil M/n \rceil$ objects [3].

Fig 2.1 Pigeonhole Principle
(Taken from Rinaldi Munir Webpage)

In password security, the pigeonholes represent the unique password human actually choose while the pigeons represent the other users. Because of human selection patterns when making a password, this drastically limit the variety of password used. Therefore, making duplicate password and system vulnerability.

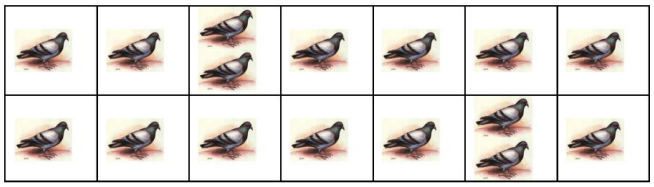


Fig 2.2 14 pigeonholes and 16 pigeons
(Taken from Rinaldi Munir Webpage)

Example if a company has 100.000 employees and the system requires an 8-character long password using lowercase letters and digits. The total possible combination is $36^8 \approx 2.82 \times 10^{12}$ with 2.8 trillion pigeonholes and only 100.000 pigeons, the theoretical probability of two employees choosing the exact same password is almost zero.

E. Stars and Bars

The Stars and Bars theorem is a method used to count how many ways a fixed number of items can be group into categories. In password security, it is used to measure how humans use characters like letters, numbers, and symbol across the length of the password. The formula is expressed as:

$$C(n + r - 1, r)$$

Example a password that require users to make an 8-character password with at least one lowercase letter (L), one capital letter (U), one number (N), and one special character (S). In this example, 8 password position as stars and 4-character types as bars. Since we need at least one of each type, we allocate 1 character to each bar, leaving 4 remaining position to distribute.

$$C(Remaining\ Characters + Bars - 1, Bars - 1)$$
$$= \frac{(4 + 4 - 1)!}{(4 - 1)! \times 4!}$$
$$= \frac{7!}{3! \times 4!}$$
$$= 35$$

35 structural profiles (e.g., 3 letters, 2 capitals, 2 numbers, 1 symbol). This theorem proves the distribution of character types, which is only a fraction of the final permutation space.

III. Combinatorial Analysis of Passwords

A. The Theoritical Password Space

A theoretically secure password assumes every character is chosen randomly from the entire available character set. Assume the least amount of characters for the minimum security is a standard 8-character password.

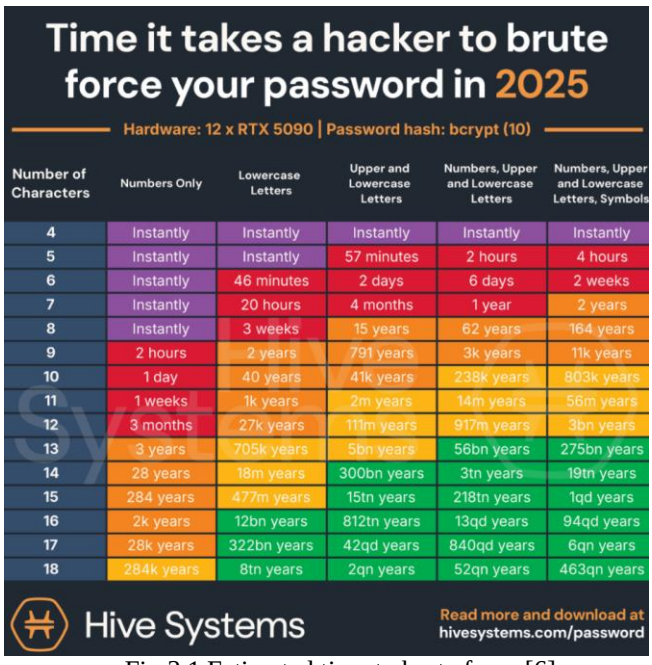
The standard character (N) typically includes:

- Lowercase letters : 26
- Capital letters : 26
- Digits : 10
- Special characters (standard US keyboard) : 32
- Total Size : 94

Using the Rule of Product, the theoretical password space for an 8-character password is:

$$S_t = 94^8 \approx 6.09 \times 10^{15}$$

possible combinations.



Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	57 minutes	2 hours	4 hours
6	Instantly	46 minutes	2 days	6 days	2 weeks
7	Instantly	20 hours	4 months	1 year	2 years
8	Instantly	3 weeks	15 years	62 years	164 years
9	2 hours	2 years	791 years	3k years	11k years
10	1 day	40 years	41k years	238k years	803k years
11	1 weeks	1k years	2m years	14m years	56m years
12	3 months	27k years	111m years	917m years	3bn years
13	3 years	705k years	5bn years	56bn years	275bn years
14	28 years	18m years	300bn years	3tn years	19tn years
15	284 years	477m years	15tn years	218tn years	1qd years
16	2k years	12bn years	812tn years	13qd years	94qd years
17	28k years	322bn years	42qd years	840qd years	6qn years
18	284k years	8tn years	2qn years	52qn years	463qn years

Fig 3.1 Estimated time to brute force [6]

(Taken from https://www.hivesystems.com/blog/are-your-passwords-in-the-green)

B. The Mathematical Impact of Constraints

Suppose a system has a rule where a password must be exactly 8-characters long and contain at least one digit. Rather than creating a space of 94^8, we must subtract the complement (password without digits). Therefore, the valid space is $94^8 - 84^8 \approx 3.61 \times 10^{15}$. While it looks secure, these restrictive rules can give attackers a mathematical guarantee about the password structure, making mask attacks possible.

C. The Effective Password Space From Human Patterns

Most humans are incapable of making and memorizing random 8 characters password, especially if they have a lot of accounts spread all over other apps and websites. Therefore, they fall back on patterns and words they can memorize easily like their names, their birthday, or their favorite word and numbers. A common password involves:

1. A standard dictionary word (assuming it is 6-letter long)
2. Capitalizing the first letter
3. Adding 2 digits at the end

We can calculate the effective space of this specific human pattern using combinatorics using the Rule of Product. Assuming the English dictionary has roughly 100.000 6-letter words:

- Total common word : 100.000 possibilities
- Capitalizing the first letter : 1 possibilities
- 2 digits at the end (00 to 99) : 100 possibilities

$$S_e = 100{,}000 \times 1 \times 100 = 10{,}000{,}000 = 10^7$$

As we can see, the gap between theoretical space and the effective space of human patterns is huge. In this case, hackers can easily and quickly exploit our passwords using dictionary and mask attacks, rendering the theoretical math irrelevant against real-world human behavior.

D. *The "Leetspeak" Illusion*

When users are forced to make a complex password, a common behavior is to use "leetspeak" or in other words substituting a number or character instead of writing a whole word with letters. For example, instead of using the word "password", a user might type "P@55w0rd!". To an average person, this might feel more secure because it satisfies capital, lowercase, number, and symbol requirements.

However, if we use combinatorial perspective, this barely increases the effective search space. Modern hackers are fully aware of these behavior. Current brute-force dictionaries already include these common substitutions as built-in rules (e.g., replacing every 'a' with '@' or '4'). If a base dictionary contains 100.000 words, applying a standard set of 10 substitution rules only expand the search space a little from 100.000 to 1.000.000 possible combinations. Mathematically, substituting characters does not create true randomness, it creates a slightly larger, but still a predictable pattern.

E. *Pigeonhole Principle in Database Leaks*

Assume there is a localized corporate network with 50.000 employees. If 80% of these employees use the predictable pattern calculated above (10^7 effective combinations), we can apply the Pigeonhole Principle. While 10^7 pigeonholes are mathematically larger than 50.000 pigeons, human patterns can shrink the distribution heavily toward a fraction of those words (e.g., a company name, local sport teams, "Password12"). If most of the employees choose from the same 250 common words, there are only $250 \times 100 = 25{,}000$ total password combinations (pigeonholes). Because the 40,000 employees (pigeons) outnumber the 25,000 available combinations, the Pigeonhole Principle mathematically guarantees that multiple employees share the exact same password, allowing hackers to break into their systems

IV. Program Simulation & Analysis

To prove the mathematical gap calculated in Section III, the author made a simulation developed using Python. By mapping the possible combinations for different password types, the program estimates the time needed to successfully execute a brute-force attack. The program assumes a processing capacity of 100 billion (10^{11}) offline guesses per second, reflecting the performance of modern GPU-based cracking systems.

A. *Experimental Python Code*

The author made the following Python program implementation that calculates the worst case scenario crack times based on the combinatorial spaces.

```python
def calculate_crack_time(combinations, guesses_per_second):
    seconds = combinations / guesses_per_second
    if seconds < 60: return f"{seconds:.4f} seconds"
    elif seconds < 3600: return f"{seconds/60:.2f} minutes"
    elif seconds < 86400: return f"{seconds/3600:.2f} hours"
    else: return f"{seconds/86400:.2f} days"

#Theoretical Random Password (8 chars, 94 alphabet)
alphabet_size = 94
length = 8
theoretical_space = alphabet_size ** length

# Human Pattern (Dictionary word + 2 digits)
dictionary_words = 100000 # 6-letter words
digits_space = 10 ** 2
human_space = dictionary_words * digits_space

# Hardware Speed: 100 Billion guesses/sec
speed = 100_000_000_000

print(f"Theoretical Space: {theoretical_space:,} combinations")
print(f"Crack Time: {calculate_crack_time(theoretical_space, speed)}")
print(f"Human Pattern Space: {human_space:,} combinations")
print(f"Crack Time: {calculate_crack_time(human_space, speed)}")
```

B. *Result and Discussion*

Simulation Output :

```
Theoretical Space: 6,095,689,385,410,816 combinations
Crack Time: 16.93 hours
Human Pattern Space: 10,000,000 combinations
Crack Time: 0.0001 seconds
```

Password Structure	Combinatorial Space	Estimated Crack Time
8-character Theoretical Random	6.095.689.385.410.816	16.93 Hours
Human Pattern (word and digits)	10.000.000	0.0001 Seconds

12-character Theoretical Random	$\approx 4.75 \times 10^{23}$	1.5×10^{5} Years
Human Pattern (Repeated Word)	100.000	< 0.0001 Seconds

Table 4.1.Brute-Force Time Estimations (100B guesses/second)

Based on this experiment, we can understand that the security of password human made is very low because of human memory constraints. While a mathematically random 8-character password need nearly 17 hours of resistance against a high-end GPU cluster, but a patterned password with the exact same length is easily and quickly crack in less than a fraction of a millisecond.

Combinatorics shows that requiring one capital letter and a number does not actually make passwords as secure as 94^8 possibilities, instead it limits passwords to much easier as just 10^7predictable combinations. Hackers exploit this exact mathematical discrepancy by utilizing targeted dictionaries and rule-based mask attacks rather than pure brute-force iteration.

C. Practical Solution Using Passphrase

If human memory struggles to remember and memorize random generated characters, the most effective solution is by utilizing "passphrases". A passphrase consist of several random and unrelated dictionary words that is combine together into one phrase (example: "battery horse correct staple"). We can prove the effectiveness by using the Rule of Product. Instead of picking characters from the alphabet, the user picks words from the dictionary. Assume a user randomly choose four words from a list of total 10.000 common words, the combinatorial space is:

$$Total = 10{,}000 \times 10{,}000 \times 10{,}000 \times 10{,}000$$
$$= 10{,}000^4$$
$$= 10^{16} \text{ combinations}$$

This creates a very large search space of 10 quadrillion possibilities. The password avoids human biases of grammar or popular phrases because the words are chosen randomly. Not only it needs longer time and more possible combinations, a 4-word passphrase is significantly easier for a human brain to visualize and memorize than a string of random symbols.

Here is the updated Python program that calculates the search space and time to brute force a 4-word passphrase:

```python
def calculate_crack_time(combinations, guesses_per_second):
    seconds = combinations / guesses_per_second
    if seconds < 60:
        return f"{seconds:.4f} seconds"
    elif seconds < 3600:
        return f"{seconds/60:.2f} minutes"
    elif seconds < 86400:
        return f"{seconds/3600:.2f} hours"
    elif seconds < 31536000:
        return f"{seconds/86400:.2f} days"
    else:
        return f"{seconds/31536000:.2f} years"

theoretical_space = 94 ** 8

human_space = 100000 * (10 ** 2)

passphrase_space = 10000 ** 4

speed = 100_000_000_000

print(f"Theoretical Space: {theoretical_space:,} combinations")
print(f"Crack Time: {calculate_crack_time(theoretical_space, speed)}\n")

print(f"Human Pattern Space: {human_space:,} combinations")
print(f"Crack Time: {calculate_crack_time(human_space, speed)}\n")

print(f"Passphrase Space: {passphrase_space:,} combinations")
print(f"Crack Time: {calculate_crack_time(passphrase_space, speed)}")
```

Simulation Output:

```
Theoretical Space: 6,095,689,385,410,816 combinations
Crack Time: 16.93 hours

Human Pattern Space: 10,000,000 combinations
Crack Time: 0.0001 seconds

Passphrase Space: 10,000,000,000,000,000 combinations
Crack Time: 1.16 days
```

Password Structure	Combinatorial Space	Estimated Crack Time
8-Character Theoretical Random	6.095.689.385.410.816	16.93 Hours
Human Pattern	10.000.000	0.0001 Seconds
4-Word Passphrase	10.000.000.000.000.000	1.16 Days
12-Character Theoretical Random	$\approx 4.75 \times 10^{23}$	1.5×10^{5} Years

Table 4.2.Brute-Force Time Estimations With Passphrase(100B guesses/second)

V. CONCLUSION

Combinatorial analysis provides as a mathematical tool for evaluating the security of not just passwords, but also our personal information. By applying the fundamentals of counting principles, permutations, combinations, inclusion and exclusion, pigeonhole, and stars and bars, we can calculate the theoretical strength of security systems.

However, there is a massive gap between theoretical password security compared to password from human selection patterns which drastically reduce the actual combinatorial space, making password easier to predict and break into. Therefore, moving forward, security systems must be designed to adapt human behavior rather than relying on mathematical ideals that users cannot achieve. Instead, systems should encourage mathematical alternatives like multi-word passphrases, while utilizing two-factor authentication to secure the inevitable vulnerabilities of human memory.

VI. Acknowledgment

The author would like to thank God for the guidance throughout the learning process. Deepest gratitude to Bapak Dr.Ir.Rinaldi Munir, MT. for providing the foundational course materials, references that made the analysis in this paper possible.

References

[1] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *2012 IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, 2012, pp. 538–552. doi: 10.1109/SP.2012.49.

[2] R. Munir, "Kombinatorika (Bagian 1)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika STEI-ITB, 2026.

[3] R. Munir, "Kombinatorika (Bagian 2)," Bahan Kuliah IF1220 Matematika Diskrit, Program Studi Teknik Informatika STEI-ITB, 2026.

[4] M. Awad, Z. Al-Qudah, S. Idwan, and A. H. Jallad, "Evaluating Password Behavior at a Small University," *Journal of Computer Science*, vol. 15, no. 1, pp. 1-9, Jan. 2019. doi: 10.3844/jcssp.2019.1.9.

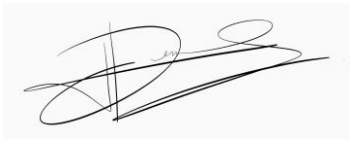
[5] Champlain College, "When is a Password Good Enough?," Champlain College Online. [Online]. Available: https://online.champlain.edu/blog/when-password-good. [Accessed: 18-Jun-2026].

[6] Hive Systems, "Are Your Passwords in the Green?," Hive Systems Blog. [Online]. Available: https://www.hivesystems.com/blog/are-your-passwords-in-the-green. [Accessed: 18-Jun-2026].

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, 18 Juni 2026



Denzel Santoso
13525014