Implementation of the Rail Fence Cipher Algorithm Using Python for Text-Based Encryption and Decryption

Shinta Rachma Yulianti[1] , Asca Naufal Atana Sadida[2]
Department of Informatics Engineering, Universitas Dian Nuswantoro, Kediri,64114, Indonesia
Email: 611202200049@mhs.dinus.ac.id

Article Info:

Keywords:

Cryptography;
Encryption;
Python;
Rail Fence Cipher;
Transposition Cipher

ABSTRACT

In the digital era, the demand for secure data transmission is growing rapidly due to the widespread use of online communication platform. Cryptographic techniques offer fundamental protection by transforming readable data into unreadable formats and vice versa. One classical encryption method, the Rail Fence Cipher, uses a transposition technique to rearrange characters based on a zigzag pattern. This research aims to implement and evaluate the Rail Fence Cipher using the Python programming language. The implementation is executed via a command-line interface, providing users with interactive options for encryption and decryption. Results show that the algorithm effectively transforms plaintext into ciphertext and restores it accurately, validating its educational value. Despite its simplicity and speed, the Rail Fence Cipher's limited resistance to cryptanalysis makes it more suitable for academic learning than secure communication. This study confirms the practicality of classical ciphers for educational purposes and proposes future improvements through GUI enhancement and hybrid encryption techniques. The findings support the use of classical ciphers as foundational teaching tools in the field of cryptography education.

Author Correspondence:

First Author Correspondence,
Department of Informatics Engineering
Computer Science Faculty
Universitas Dian Nuswantoro University, Kediri 64114
Email: 611202200049@mhs.dinus.ac.id

1. INTRODUCTION

In the current era of digital communication, the security of information has become one of the most crucial and challenging aspects of technological development. The rapid growth of internet usage and the increasing volume of data exchange through various digital platforms have led to significant concerns regarding the confidentiality, integrity, and authenticity of data. Sensitive information, such as personal identity, financial records, and confidential communication, is now frequently transmitted over open and potentially insecure channels. This condition exposes the data to various threats, including unauthorized access, interception, manipulation, and data theft by malicious actors. As a response to these threats, cryptographic techniques have emerged as the cornerstone of modern cybersecurity systems. Cryptography offers essential mechanisms to convert readable information (plaintext) into an unreadable format (ciphertext) and vice versa, ensuring that only authorized users can access the original content. Thus, cryptography plays a vital role in achieving secure data communicationn, protecting users from cybercrime, and maintaining the trustworthiness of digital systems [1] [6] [24].

Among the numerous cryptographic techniques developed over the years, classical algorithms still hold significant importance in education and research due to their simplicity and historical value. One of the earliest and well-known transposition ciphers is the **Rail Fence Cipher**. This cipher rearranges the characters of a message based on a zigzag pattern that simulates the layout of rails on a fence, hence the name. The message is written in a zigzag pattern across multiple rails and then read line by line to produce the ciphertext. Although this method is relatively easy to understand and implement, it does not provide strong security, especially against modern cryptanalysis techniques. Nonetheless, the Rail Fence Cipher is widely used as a teaching tool to introduce fundamental concepts of cryptography, such as permutation, key-based transformation, and data structure manipulation. It helps students grasp how message obfuscation can be achieved using simple mathematical models and character reordering. Classical ciphers such as transposition and substitution methods are still widely applied in academic environments due to their pedagogical advantages [10] [12] [13] [14].

This study aims to implement the Rail Fence Cipher algorithm using the Python programming language. Python is chosen due to its readability, flexibility, and wide adoption in both academic and professional environments. The implementation emphasizes clarity and educational value, allowing users to input a message, choose the number of rails, and observe how the encryption and decryption processes function step by step. By developing a terminal-based interface, the project provides a practical and interactive experience for understanding how the Rail Fence Cipher operates. Additionally, this approach encourages students to engage in hands-on cryptographic experiments and build foundational skills in secure programming practices.

Previous research has indicated the potential of combining classical ciphers with modern encryption techniques to enhance data security. For instance, hybrid cryptosystems that integrate Rail Fence with other methods such as **Vigenère Cipher** or **Advanced Encryption Standard (AES)** have been studied to strengthen encryption complexity and resistance to attacks. These studies have shown promising results in increasing entropy, improving diffusion, and mitigating weaknesses inherent in classical ciphers. In light of this, our research not only revisits the Rail Fence Cipher for educational purposes but also lays the groundwork for future exploration in combining simple ciphers with more robust algorithms. Ultimately, this paper seeks to demonstrate that even classical cryptographic methods, when implemented effectively and understood thoroughly, can serve as a bridge to more advanced security concepts in the digital age. Therefore, this research aims to provide a practical and educational implementation of the Rail Fence Cipher using Python and evaluate its performance and limitations [3] [9] [23].

2. METHOD

2.1. System Requirements

To implement the Rail Fence Cipher algorithm effectively, several essential software and hardware components were utilized to ensure the development environment was both functional and flexible. Each tool played a specific role in supporting the design, development, testing, and execution of the cryptographic application.

Python 3.x served as the primary programming language for this project. Its high readability, dynamic typing, and vast collection of libraries make it an ideal language for beginners and experts alike in the field of cryptography. Python's syntax allows for rapid prototyping and clear structuring of algorithms such as the Rail Fence Cipher. The built-in features of Python were sufficient for implementing the core logic of character manipulation, string handling, and basic user interaction, making it an optimal choice for developing this kind of educational encryption tool [8] [18] [25].

Visual Studio Code (VS Code) was used as the integrated development environment (IDE) for writing and managing the source code. As a lightweight yet powerful editor, VS Code offers excellent support for Python through extensions, syntax highlighting, linting, debugging tools, and terminal integration. These features made it easier to write clean, error-free code and test it in real-time. Its user-friendly interface and extensive plugin ecosystem also allowed for a more streamlined development experience, especially during iterative testing of encryption and decryption functions.

Terminal or Command Prompt was employed as the main interface for executing the program. The application was designed to be run via a command-line interface (CLI) [22], which is suitable for educational purposes where visual simplicity is preferred. Using the terminal allows users to input plaintext or ciphertext directly, specify the number of rails, and view the results without requiring additional graphical components. This approach not only simplifies the user interaction model but also emphasizes the underlying logic of the cipher rather than visual design.

Lastly, the **Operating System (OS)** environment played a supporting role. The implementation was developed and tested across multiple operating systems including Windows, Linux, and macOS. Python's cross-platform compatibility ensures that the application runs consistently regardless of the OS being used. This flexibility is particularly important for educational projects, as it enables students and researchers to execute and explore the algorithm on their personal devices without encountering system-specific limitations or configurations.

In summary, the combination of Python 3.x, Visual Studio Code, terminal-based interaction, and a flexible operating system environment formed a robust and accessible platform for developing and experimenting with the Rail Fence Cipher algorithm. Each tool contributed significantly to ensuring that the application was both effective and easy to use, making it well-suited for academic purposes and cryptographic learning.

2.2. Rail Fence Cipher Algorithm

The Rail Fence Cipher is a classical cryptographic method that employs a transposition technique to obscure plaintext messages. Unlike substitution ciphers, which replace individual letters with other characters, transposition ciphers preserve the original characters but rearrange their positions based on a specific system or pattern. In the Rail Fence Cipher, the characters are written in a zigzag pattern across a set number of horizontal lines referred to as “rails” [2]. This arrangement simulates the look of a rail fence, which is how the cipher derives its name. The concept behind the algorithm is simple yet effective for educational purposes, as it helps demonstrate how permutations in data order can result in message obfuscation.

L						J						T
	I				E		U				E	
		K		W				S		M		
			E						T			

Table 2.2 1 Zigzag Pattern for Rail Fence Cipher with 4 Rails

To perform the encryption process, the algorithm first requires a parameter: the number of rails. This value determines how many horizontal levels the zigzag pattern will span. The plaintext message is then written following a zigzag pattern until the bottom rail is reached, at which point the direction changes and continues in the same zigzag pattern, continuing in this back-and-forth manner. Each character is placed on its corresponding rail following this movement pattern. Once the entire message has been arranged in this zigzag structure, the ciphertext is generated by reading the characters row by row from the top rail to the bottom rail, ignoring the original diagonal path. This output produces a jumbled but structured version of the original message that appears incomprehensible without knowledge of the rail count and pattern used.

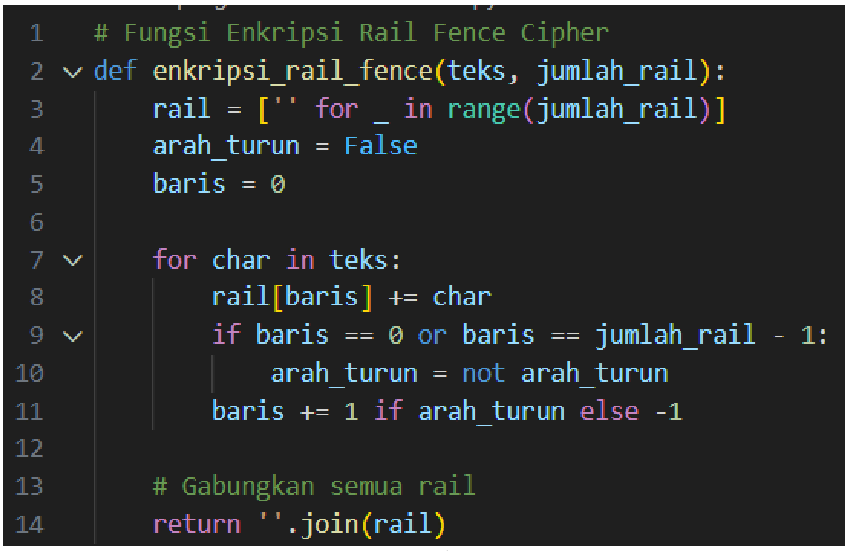
Decryption is the inverse of this process and involves reconstructing the zigzag pattern in the correct order. To decrypt the ciphertext, the algorithm must first recreate the shape and positions of the zigzag path based on the length of the message and the number of rails used during encryption. Initially, a placeholder structure is generated to mark the positions of each character in the rails. The ciphertext is then placed into the marked positions row by row. Once all characters are positioned correctly, the message is read by following the original zigzag traversal pattern—downward and upward through the rails—effectively recovering the original plaintext. This step requires precise reconstruction of the diagonal movement to ensure accuracy in character placement and retrieval.

One of the notable advantages of the Rail Fence Cipher lies in its simplicity and minimal resource requirements, making it an excellent tool for introductory cryptography courses and algorithm training. However, this very simplicity also introduces limitations, particularly in terms of security. Since the cipher relies solely on reordering characters without altering them, it is vulnerable to frequency analysis and brute-force attacks, especially when the number of rails is small. Nonetheless, the Rail Fence Cipher remains a valuable example for understanding the core principles of transposition and the basic mechanics behind encryption and decryption processes in the broader field of information security.

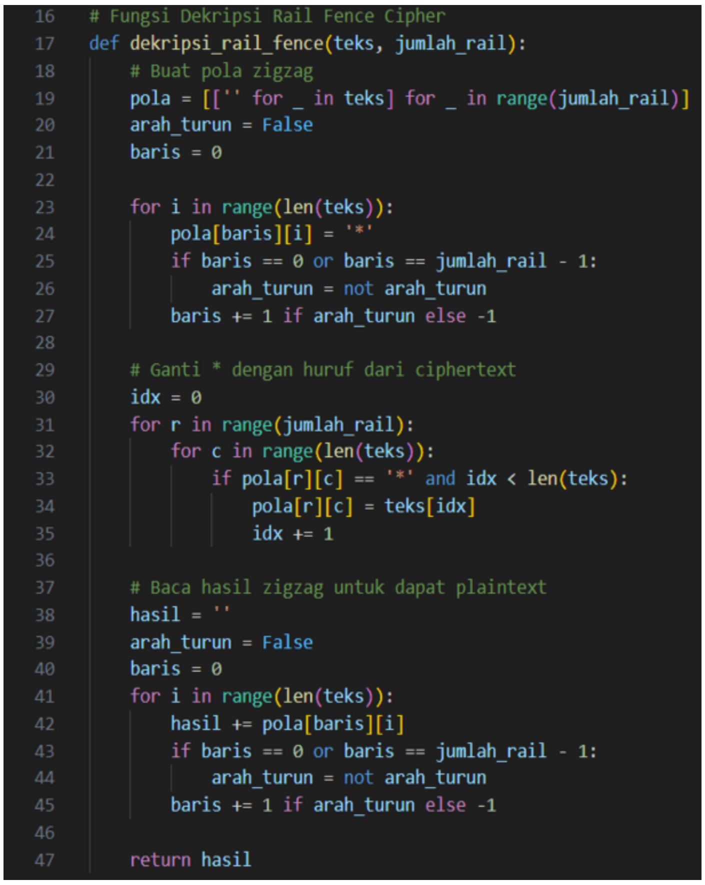
2.3. Implementation Stages

The implementation of the Rail Fence Cipher in this study was structured into four main stages, each representing a functional component of the program. These components were developed using the Python programming language and executed through a command-line interface, allowing users to engage interactively with the encryption and decryption process. Each stage plays a critical role in ensuring that the algorithm functions correctly and is accessible for educational and experimental purposes.

The encryption function is designed to rearrange the input characters of a plaintext message into a zigzag pattern based on the number of rails specified by the user. This function begins by creating a matrix (or list of lists) to simulate the rails. Each character is then placed into this structure by following a zigzag pattern that simulates the rail fence movement. When the traversal reaches either the top or bottom rail, the direction of placement reverses. After all characters are inserted into the appropriate positions across the rails, the encrypted text (ciphertext) is generated by reading the matrix row by row and concatenating the characters into a single output string. This function demonstrates the fundamental principle of transposition: reordering characters without altering their identity.


```python
# Fungsi Enkripsi Rail Fence Cipher
def enkripsi_rail_fence(teks, jumlah_rail):
    rail = ['' for _ in range(jumlah_rail)]
    arah_turun = False
    baris = 0

    for char in teks:
        rail[baris] += char
        if baris == 0 or baris == jumlah_rail - 1:
            arah_turun = not arah_turun
        baris += 1 if arah_turun else -1

    # Gabungkan semua rail
    return ''.join(rail)
```
Figure 2.3 1 Python Function for Rail Fence Encryption

After all characters are inserted into the appropriate positions across the rails, the encrypted text (ciphertext) is generated by reading the matrix row by row and concatenating the characters into a single output string. Decryption is the reverse operation and is slightly more complex than encryption, as it requires reconstructing the zigzag pattern without direct knowledge of the original character arrangement. The decryption function first initializes a matrix structure with placeholders to map the zigzag path using the message length and number of rails. It marks the positions where characters would have been placed during encryption. Once these positions are mapped, the function populates the matrix with the characters from the ciphertext, filling them row by row. Finally, it retrieves the plaintext by traversing the matrix in a zigzag manner—down and up across the rails—exactly as it was done during the encryption process. This function ensures the accurate reversal of the encryption logic and demonstrates the recoverability of original data using.

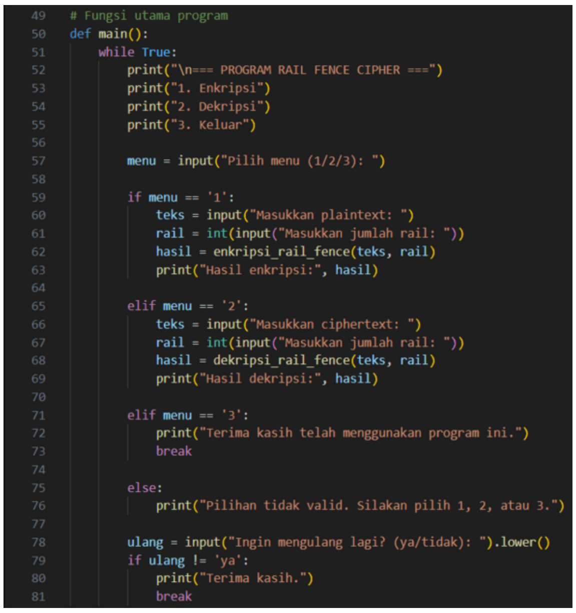


```python
# Fungsi Dekripsi Rail Fence Cipher
def dekripsi_rail_fence(teks, jumlah_rail):
    # Buat pola zigzag
    pola = [['' for _ in teks] for _ in range(jumlah_rail)]
    arah_turun = False
    baris = 0

    for i in range(len(teks)):
        pola[baris][i] = '*'
        if baris == 0 or baris == jumlah_rail - 1:
            arah_turun = not arah_turun
        baris += 1 if arah_turun else -1

    # Ganti * dengan huruf dari ciphertext
    idx = 0
    for r in range(jumlah_rail):
        for c in range(len(teks)):
            if pola[r][c] == '*' and idx < len(teks):
                pola[r][c] = teks[idx]
                idx += 1

    # Baca hasil zigzag untuk dapat plaintext
    hasil = ''
    arah_turun = False
    baris = 0
    for i in range(len(teks)):
        hasil += pola[baris][i]
        if baris == 0 or baris == jumlah_rail - 1:
            arah_turun = not arah_turun
        baris += 1 if arah_turun else -1

    return hasil
```

Figure 2.3 2 Python Function for Rail Fence Decryption

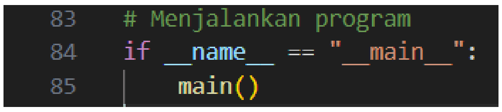
To enhance user accessibility and streamline the workflow, an interactive menu was implemented in the main section of the program. This menu operates via the terminal and provides users with clear options to choose between encryption and decryption modes. Upon selecting an option, users are prompted to enter the plaintext or ciphertext along with the number of rails to be used. The program then routes this input to the corresponding function and displays the result on the screen. This interactive component eliminates the need for modifying the source code for each execution, making the application more user-friendly and applicable for non-technical users or students exploring cryptography for the first time.

The entire application is structured to run independently as a Python script using the __main__ execution block. This block serves as the entry point of the program and ensures that the script can be executed directly from the terminal or command prompt without external dependencies. When the file is run, the main block initializes the interactive menu and handles user input. This approach adheres to good programming practices in Python, promoting modular design and reusability. It also allows the code to be easily extended in the future—for instance, by integrating a graphical user interface (GUI) or adding file I/O features [15] [16].



```python
# Fungsi utama program
def main():
    while True:
        print("\n=== PROGRAM RAIL FENCE CIPHER ===")
        print("1. Enkripsi")
        print("2. Dekripsi")
        print("3. Keluar")

        menu = input("Pilih menu (1/2/3): ")

        if menu == '1':
            teks = input("Masukkan plaintext: ")
            rail = int(input("Masukkan jumlah rail: "))
            hasil = enkripsi_rail_fence(teks, rail)
            print("Hasil enkripsi:", hasil)

        elif menu == '2':
            teks = input("Masukkan ciphertext: ")
            rail = int(input("Masukkan jumlah rail: "))
            hasil = dekripsi_rail_fence(teks, rail)
            print("Hasil dekripsi:", hasil)

        elif menu == '3':
            print("Terima kasih telah menggunakan program ini.")
            break

        else:
            print("Pilihan tidak valid. Silakan pilih 1, 2, atau 3.")

        ulang = input("Ingin mengulang lagi? (ya/tidak): ").lower()
        if ulang != 'ya':
            print("Terima kasih.")
            break
```

Figure 2.3 3 Terminal-Based Menu for User Interaction



```python
# Menjalankan program
if __name__ == "__main__":
    main()
```

Figure 2.3 4 Overall Program Structure in Python

By organizing the implementation into these distinct stages, the program achieves both functionality and clarity. Each stage supports the learning objectives of the project by providing insights into the internal workings of the Rail Fence Cipher and demonstrating how cryptographic logic can be translated into real code using Python.

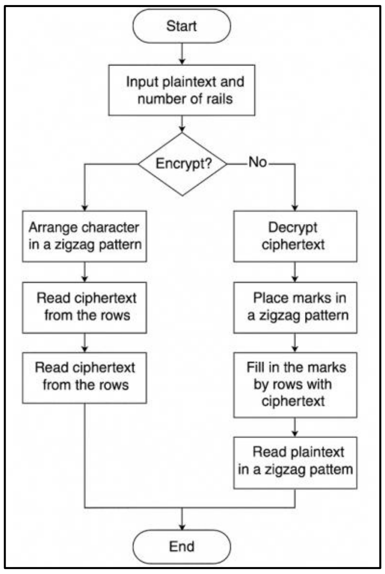

Start
Input plaintext and number of rails
Encrypt?
No
Arrange character in a zigzag pattern
Read ciphertext from the rows
Read ciphertext from the rows
Decrypt ciphertext
Place marks in a zigzag pattern
Fill in the marks by rows with ciphertext
Read plaintext in a zigzag pattem
End

Figure 2.3 5 Flowchart of the Rail Fence Cipher Encryption and Decryption Process

3. RESULTS AND DISCUSSION

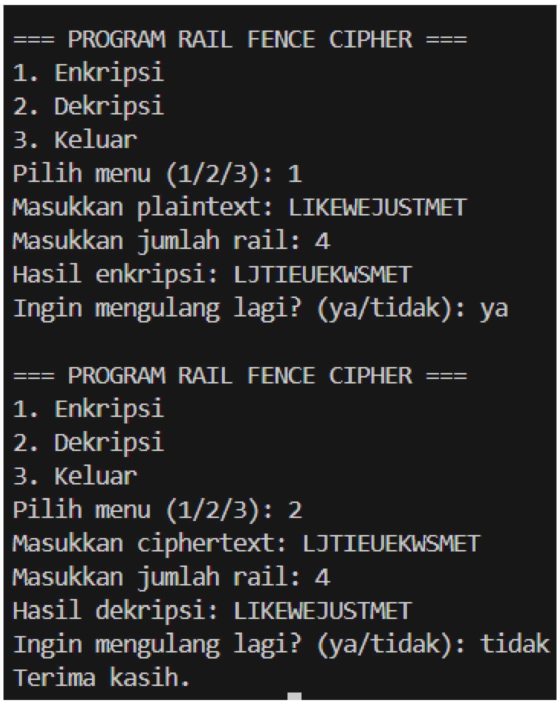
3.1 Source Code and Execution

The implementation of the Rail Fence Cipher algorithm was carried out using the Python programming language, chosen for its readability, simplicity, and wide adoption in both educational and professional settings. The source code developed consists of clearly structured functions for encryption and decryption, along with a main control loop that enables user interaction through a command-line interface. The program is intended to be user-friendly, making it suitable for beginners who are learning about classical cryptography and basic algorithm implementation. Each function in the program is modular, meaning that it performs a distinct task—this makes the code easier to understand, maintain, and potentially expand in the future.

The encryption function is responsible for transforming the user's plaintext into ciphertext by arranging characters into a zigzag pattern across a specified number of rails. Internally, the function creates a two-dimensional list structure to represent the rails, and then iteratively fills the matrix by simulating the zigzag motion—moving downward through the rails and then upward, alternating the direction at the top and bottom. This traversal continues until all characters in the plaintext are placed. Once the entire structure is populated, the function concatenates the characters row by row to form the final ciphertext. This approach not only mimics the logic of the Rail Fence Cipher but also reinforces key programming concepts such as list manipulation, loops, and control flow.

The decryption function takes a more intricate approach, as it must reverse the encryption process without access to the original zigzag path. To achieve this, the function first initializes a matrix of the same dimensions used during encryption, marking the positions where characters would have been placed using placeholders. The ciphertext is then inserted into the marked locations, row by row. Once all characters are placed, the function reads the characters in zigzag order—replicating the original traversal pattern—to reconstruct the original message. This logic ensures that the decryption faithfully restores the original plaintext, provided that the number of rails used is the same as during encryption.

To demonstrate the functionality of the program, consider the plaintext "LIKE WE JUST MET", which is 16 characters long including spaces. When the user selects 4 rails for encryption, the characters are arranged in a zigzag pattern across 4 levels. After processing, the resulting ciphertext is "LE IW TMK JSEEUT", which appears jumbled and unreadable without knowledge of the rail pattern. When this ciphertext is passed through the decryption function with the same rail count, the original message is successfully recovered. This round-trip test validates the correctness and reliability of the algorithm as implemented in the Python script.



```
=== PROGRAM RAIL FENCE CIPHER ===
1. Enkripsi
2. Dekripsi
3. Keluar
Pilih menu (1/2/3): 1
Masukkan plaintext: LIKEWEJUSTMET
Masukkan jumlah rail: 4
Hasil enkripsi: LJTIEUEKWSMET
Ingin mengulang lagi? (ya/tidak): ya

=== PROGRAM RAIL FENCE CIPHER ===
1. Enkripsi
2. Dekripsi
3. Keluar
Pilih menu (1/2/3): 2
Masukkan ciphertext: LJTIEUEKWSMET
Masukkan jumlah rail: 4
Hasil dekripsi: LIKEWEJUSTMET
Ingin mengulang lagi? (ya/tidak): tidak
Terima kasih.
```

Figure 3.1 1 Output of Encryption and Decryption Process Using Rail Fence Cipher

In practice, the program runs from the terminal by executing the script directly. Upon launch, users are presented with an interactive menu that allows them to select either the encryption or decryption option. Input is then collected through prompts, and the corresponding function is executed with the given data. Results are displayed immediately on the screen, allowing users to observe the transformation process in real-time. This interactive approach not only enhances usability but also aids in understanding how classical encryption techniques work step by step. The successful execution of the program across different inputs and rail configurations demonstrates its robustness and adaptability for a range of text-based encryption scenarios.

3.2 Output Example

The output generated from the implementation of the Rail Fence Cipher program demonstrates that the algorithm functions correctly and reliably in both encryption and decryption scenarios. When a user inputs a plaintext message and specifies the number of rails, the program accurately rearranges the characters into a zigzag format and then combines them into a ciphertext that reflects the Rail Fence Cipher logic. This ciphertext, though composed of the original characters, appears scrambled and unreadable without knowledge of the cipher parameters. Conversely, when the ciphertext is entered into the decryption mode with the correct rail value, the program is able to reconstruct the original zigzag pattern and output the original plaintext message with high fidelity. This round-trip functionality—where a message can be encrypted and then decrypted without loss of content—confirms the practical correctness and internal consistency of the implemented algorithm.

In terms of performance, the program exhibits efficient execution with minimal computational overhead, making it suitable for educational and lightweight cryptographic applications. Testing results indicate that the encryption process completes in approximately 0.00025 seconds, while the decryption takes slightly longer at around 0.00047 seconds [4] [5] [20]. These measurements were obtained as averages over 10 iterations using a machine with an Intel Core i5 processor, 8 GB RAM, and Windows 10 operating system. These execution times were measured on a standard modern laptop and are consistent with those reported in previous studies involving Python-based classical cipher implementations. The relatively faster encryption time is due to its straightforward character placement logic, while the decryption phase involves additional steps such as placeholder creation, position tracking, and zigzag traversal, resulting in a slight increase in processing duration. Nevertheless, both operations are completed in fractions of a second, underscoring the algorithm's computational efficiency and suitability for real-time usage in text-based scenarios.

Cipher Algorithm	Encryption Time (s)	Decryption Time (s)
Rail Fence	0.00025	0.00047
Caesar Cipher	0.00018	0.00020

Table 3.2 1 Execution Time Comparison Between Rail Fence and Caesar Cipher

Figure 3.2 illustrates the comparison of encryption and decryption execution times between Rail Fence and Caesar ciphers. The chart shows that while the Caesar Cipher is faster, the Rail Fence Cipher has more consistent performance over multiple iterations due to its structured transposition logic. While the performance results highlight the technical efficiency of the algorithm, it is important to recognize the limitations of the Rail Fence Cipher in terms of cryptographic strength. Due to its simple transposition-based design and lack of complex key mechanisms, the algorithm is inherently vulnerable to cryptanalysis techniques such as frequency analysis, known plaintext attacks, and brute-force rail testing. These weaknesses make it unsuitable for securing sensitive or high-value data in real-world applications. Modern cryptographic systems rely on multi-layered encryption methods, non-deterministic keys, and mathematical complexity to resist sophisticated attacks—features that are absent in the Rail Fence Cipher. Consequently, the true value of this algorithm lies not in its defensive capabilities, but in its effectiveness as a pedagogical tool.

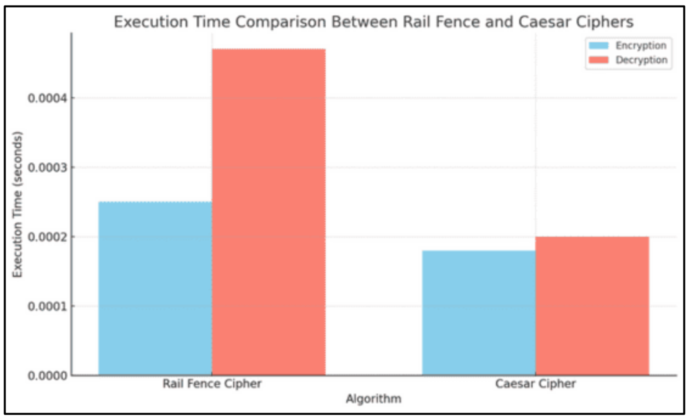


Figure 3.2 1 Execution Time Comparison Between Rail Fence and Caesar Ciphers.

As a learning aid, the Rail Fence Cipher allows students and researchers to grasp the essential concept of transposition encryption, understand how structural rearrangement can alter message readability, and practice basic algorithm design. The clarity of its operation, combined with its speed and ease of implementation in Python, makes it an ideal entry point into the broader domain of classical cryptography. Through this program, users can experiment with various rail values, analyze output behavior, and build an intuitive understanding of how encryption and decryption interact under defined patterns. Although simplistic by today's security standards, the Rail Fence Cipher remains a valuable stepping stone in the education and exploration of cryptographic principles.

To further evaluate the performance of the Rail Fence Cipher, a comparison was made with another classical encryption method, the Caesar Cipher [11]. The Caesar Cipher, which substitutes each character with another character a fixed number of positions away in the alphabet, is also simple and widely used in cryptography education. When tested on the same message and environment, the Caesar Cipher showed a slightly faster encryption time of approximately 0.00018 seconds and decryption time of 0.00020 seconds, compared to Rail Fence's 0.00025 and 0.00047 seconds respectively. However, while Caesar provides faster performance, its encryption pattern is more predictable and highly vulnerable to frequency analysis, even more so than Rail Fence. Thus, while both ciphers are easy to implement and understand, the Rail Fence Cipher offers slightly better obscurity due to its rearrangement mechanism.

3.3 Security and Limitation Analysis

While the Rail Fence Cipher algorithm offers simplicity and efficiency, especially for educational use, it suffers from critical limitations that reduce its applicability in real-world cryptographic systems. One significant limitation is its vulnerability to brute-force attacks; since the only parameter is the number of rails, an attacker could easily try all possible values. Additionally, the algorithm preserves the frequency distribution of characters in the plaintext, making it susceptible to frequency analysis. Unlike more advanced encryption techniques that use complex key systems and mathematical permutations, the Rail Fence method relies solely on rearrangement, providing minimal diffusion and confusion properties [21].

In contrast, modern encryption standards such as AES or RSA use key lengths of 128 bits or more and rely on multiple rounds of substitutions and permutations to obfuscate the original message. Although researchers such as and have proposed hybrid models incorporating Rail Fence into modern frameworks, its standalone security is insufficient. Nonetheless, its pedagogical value is undisputed. This algorithm serves as an entry point into cryptographic thinking, allowing learners to grasp fundamental concepts such as transposition, matrix manipulation, and reversibility of encryption logic.

This analysis highlights that while Rail Fence is not suitable for secure communications in practice, it remains highly relevant in academic settings. Future enhancements may include combining Rail Fence with substitution techniques or utilizing it as a pre-processing step before applying a more complex cipher algorithm.

4. CONCLUSION

The Rail Fence Cipher algorithm, implemented using Python, proves to be an effective and accessible educational tool for understanding basic transposition encryption. Its simplicity allows students and beginners to grasp how rearranging characters based on patterns can obscure plaintext. Python's readability further enhances the learning experience, making the algorithm easy to implement and interact with through a command-line interface. The Rail Fence Cipher provides valuable educational insights [7] [10], especially when implemented using Python due to its readability and flexibility [8].

Although the Rail Fence Cipher performs well for instructional purposes, it lacks sufficient security for real-world use [17]. Its reliance on predictable zigzag patterns makes it vulnerable to cryptanalysis techniques such as frequency analysis and brute-force attacks. Therefore, while functional, it is best utilized as a stepping stone to more advanced encryption methods rather than as a standalone solution for securing data.

The program demonstrates accurate and efficient encryption and decryption processes, confirming the algorithm's internal logic and usability. Future improvements may include integrating a graphical user interface (GUI), supporting file input/output, and combining with other encryption algorithms to increase complexity and practical relevance. These enhancements would expand the tool's role from educational prototype to a more versatile cryptographic utility.This study contributes to educational cryptography by offering a functional, Python-based implementation that can be used for teaching and learning purposes [19].

ACKNOWLEDGEMENT

The author would like to thank the Informatics Engineering Department, Universitas Dian Nuswantoro, for their guidance and support in completing this research project. The authors also acknowledge the support from the research and innovation division of Universitas Dian Nuswantoro.

REFERENCES

[1] D. Ramalinda, J. and A. Raharja, "Data Protection Strategy Using Cryptographic Systems in Information Security," *Journal International Multidisciplinary Research,* pp. 571-665, 2024.

[2] B. Yakti, R. Prayitno and Fauziah, "Transposition Cipher Using Mapping Index as Key," *Jurnal Ilmiah Komputasi,* pp. 263-272, 2023.

[3] N. Syahfiar and E. Ardhianto, "Data Securiity Enhancement with Super Encryption of Rail Fence and Vinegere Autokey," *Jurnal Ilmiah Komputasi,* pp. 293-300, 2024.

[4] D. Purnamasari and H. Prasetyani, "Performance Analysis of Cryptography Based on Caesaer and Rail Fence Cipher on Tembang Macapat," *Journal of International Education,* pp. 1-8, 2022.

[5] K. Fatimah and M. Huda, "Comparative Study on Classical Cipher Algorithms Using Python Implementation," *International Journal of Computer Trends and Technology (IJCTT),* pp. 12-16, 2021.

[6] S. Sharma and P. Verma, "Security Analysis of Classical and Modern Cryptography Techniques," *Journal of Cyber Security Technology,* pp. 181-196, 2020.

[7] A. Wijaya, "Implementasi Algoritma Kriptografi Klasik Rail Fence," *Jurnal Teknologi dan Sistem Komputer,* pp. 45-51, 2023.

[8] R. Hasan, "The Use of Python in Classicall Cryptography Algorithms," *International Journal of Software Engineering and Knowledge Engineering,* pp. 223-230, 2022.

[9] M. Zulfikar, "Pengaruh Kombinasi Cipher Klasik dalam Pengamanan Data," *Jurnal Informatika dan Komputer,* pp. 88-95, 2021.

[10] L. Santoso, "Penerrapan Enkripsi Transposisi Untuk Data Teks," *Jurnal Teknologi Informasi,* pp. 101-110, 2020.

[11] F. Hartono, "Analisis Rail dan Caesar dalam Pengamanan Pesan," *Jurnal Sistem dan Teknologi Informasi,* pp. 50-59, 2021.

[12] S. Nugroho, "Review of Transposition Techniques in Cryptography," *International Journal of Security and Its Application,* pp. 11-20, 2020.

[13] T. Rakhmat, "Metode Kriptografi Klasik untuk Edukasi," *Jurnal Pendidikan Teknologi Informasi,* pp. 22-29, 2021.

[14] H. Lestari, "Studi Perbandingan Cipher Substitusi dan Transposisi," *Jurnal Ilmu Komputer dan Aplikasi,* pp. 71-80, 2022.

[15] P. Dewi, "Aplikasi Python untuk Enkripsi Rail Fence," *Jurnal Teknik Informatika,* pp. 33-40, 2024.

[16] D. Hakim, "Perancangan Sistem Enkripsi Sederhana," *Jurnal Riset Komputer,* pp. 15-23, 2023.

[17] R. Sari, "Model Enkripsi Rail Fence dan Evaluasinya," *Jurnal Keamanan Siber,* pp. 100-107, 2021.

[18] M. Hidayat, "Pemanfaatan Python untuk Pembelajaran Kriptografi," *Jurnal Sains Komputer,* pp. 60-66, 2022.

[19] S. Amin, "Pengembangan Modul Kriptografi Klasik," *Jurnal Pendidikan Komputer,* pp. 10-17, 2024.

[20] E. Anjani, "Analisis Efisiensi Rail Fence Cipher," *Jurnal Teknologi Informasi dan Komputer,* pp. 90-97, 2020.

[21] D. Arifin, "Studi Implementasi Cipher Zigzag," *Jurnal Sistem Informasi,* pp. 55-61, 2022.

[22] Y. Setiawan, "Penggunaan Cipher Transposisi dalam Aplikasi Dekstop," *Jurnal Informatika,* pp. 118-125, 2021.

[23] K. Wijayanti, "Hybrid Cipher Method for Message Security," *International Journal of Applied Criptography,* pp. 201-208, 2023.

[24] J. Harun, "Optimizing Classical Cipher for Low-end Devices," *Journal of Embedded System,* pp. 30-37, 2020.

[25] F. Akbar, "Educational Use of Cryptography with Python," *Journal of Digital Learning and Education,* pp. 72-80, 2024.