**SIMAD UNIVERSITY**

****

**A Novel Caesar Cipher Variant Algorithm to Prevent Brute Force Attacks**

**GROUP 1**

**Information Security**

**FACULTY OF COMPUTING**

**SIMAD UNIVERSITY**

**Eng. Adam Mohamed Muhudin**

**JULY 2025**

# **Abstract**

This report proposes a novel encryption algorithm based on a customized Caesar Cipher mechanism enhanced to resist brute force attacks. The technique incorporates a repeating 3-letter key, symbol and number noise filtering, and dynamic shifting based on character positions. Unlike traditional Caesar Cipher methods with fixed shift values, this approach leverages character-by-character mapping using key repetition, which dramatically increases the complexity of brute force decryption. Additionally, the encryption algorithm ignores non-alphabetic characters during the transformation process, reintroducing them only in the final output, thus preserving the original formatting. The proposed method ensures improved security while maintaining simplicity in both encryption and decryption processes.

## Table of Contents

1. Introduction
2. Problem Statement
3. Objectives
4. Overview of Caesar Cipher
5. Brute Force Vulnerabilities in Caesar Cipher
6. Proposed Algorithm – Overview
7. Encryption Methodology
8. Decryption Methodology
9. Noise Filtering Technique
10. Key Repetition Mechanism
11. Pseudocode for Encryption
12. Pseudocode for Decryption
13. Implementation Example
14. Security Analysis
15. Performance and Efficiency
16. Advantages and Limitations
17. Future Enhancements
18. Conclusion
19. References
20. Appendices

# 1. Introduction

The Caesar Cipher is one of the earliest known encryption techniques, named after Julius Caesar, who used it to protect his military communications. It works by shifting each letter of the plaintext by a fixed number of positions in the alphabet. Despite its historical significance, the Caesar Cipher is highly vulnerable to brute force attacks due to its limited number of possible key values—only 25 shifts for the English alphabet. As cryptography has evolved, the need for more robust and unpredictable encryption mechanisms has become increasingly critical, especially in the digital era where brute force attacks can be executed in milliseconds by automated systems.

Traditional Caesar Cipher implementations are deterministic and predictable. Once an attacker knows that Caesar Cipher is being used, they can simply try all 25 possible shifts to break the encryption. This weakness makes the technique unsuitable for protecting sensitive information in modern-day applications. In response to these limitations, there has been a growing interest in enhancing classical encryption algorithms with additional layers of complexity to increase their resistance against attacks while maintaining computational efficiency.

This report introduces a novel variant of the Caesar Cipher that addresses its brute force vulnerability by introducing a dynamic shifting system based on a repeating three-letter key. Each character in the plaintext is individually shifted by the alphabetical value of the corresponding character in the key, which is cyclically repeated throughout the message. Additionally, this algorithm introduces a noise filtering mechanism that excludes symbols, numbers, and punctuation from the encryption and decryption processes. These non-alphabetic characters are temporarily removed during transformation and reinserted in their original positions afterward to preserve the text's structure.

The proposed method significantly increases the number of possible key combinations compared to the standard Caesar Cipher. A three-letter key composed of 26 possible letters offers 17,576 potential key combinations, making brute force attacks substantially more difficult. By introducing this dynamic and position-dependent shifting, the algorithm improves the confusion and diffusion properties required for secure cryptographic systems, without overly complicating the process or requiring complex computational resources.

In this report, we will explore the weaknesses of traditional Caesar Ciphers, detail the construction and logic behind the proposed variant, and present a full implementation guide with pseudocode, examples, and performance analysis. The aim is to demonstrate how a simple yet powerful enhancement can transform a vulnerable classical cipher into a modern, secure alternative suitable for educational and lightweight encryption applications.

# 2. Problem Statement

Cryptographic security is a fundamental requirement in digital communication, especially in environments where sensitive data must be protected from unauthorized access. Classical encryption methods, such as the Caesar Cipher, though historically important and educationally valuable, are no longer sufficient for modern security needs. The core weakness of the Caesar Cipher lies in its simplicity: each letter in the plaintext is shifted by a fixed number, leading to only 25 possible encrypted variations for the English alphabet. This limited key space makes the cipher highly susceptible to brute force attacks, where an attacker can simply try all possible shifts to reveal the original message.

Furthermore, the traditional Caesar Cipher does not account for the presence of non-alphabetic characters such as numbers, punctuation marks, and symbols. These characters are either left unencrypted or treated inconsistently, which can introduce vulnerabilities or result in loss of formatting in the encrypted message. In practical applications, such inconsistencies can make the cipher less reliable or impractical for real-world usage.

Another key limitation of the classic Caesar Cipher is its lack of variability and adaptability. Once the shift value is known or guessed, all characters in the message can be easily decrypted. The absence of a dynamic or position-based shifting mechanism means that patterns in the text can be easily identified and exploited by attackers. As a result, the Caesar Cipher fails to provide sufficient obfuscation and complexity required for secure message encryption in modern contexts.

Brute force decryption of a standard Caesar Cipher can be accomplished by both manual methods and automated scripts within seconds, rendering it ineffective against even basic computational attacks. The risk increases significantly when encrypted messages are intercepted during transmission or stored without additional safeguards. In educational or lightweight encryption use cases, there is a need for simple ciphers that go beyond basic Caesar methods without introducing high computational overhead.

This project addresses these weaknesses by proposing an enhanced Caesar Cipher algorithm. The new design aims to increase the key space, eliminate brute force feasibility, and maintain structural integrity by filtering out and preserving non-letter characters during encryption. By introducing a repeating 3-letter key, dynamic shifting, and noise filtering, the proposed system significantly increases the security and practical usability of the Caesar Cipher while retaining its foundational simplicity.

# 3. Objectives

The primary goal of this project is to design and document an enhanced Caesar Cipher algorithm that addresses the vulnerabilities of the classical version, particularly its susceptibility to brute force attacks and its limited treatment of non-alphabetic characters. The proposed algorithm introduces a repeating 3-letter key and noise filtering to make the encryption process more secure and practically useful.

To achieve this overarching goal, the project defines the following specific objectives:

3.1 Develop a Modified Caesar Cipher Algorithm:  
To construct an encryption and decryption mechanism that uses a dynamic shift value derived from a repeated 3-character key instead of a single fixed shift. Each letter in the plaintext will be encrypted by combining it with the corresponding letter in the key, creating a more unpredictable and secure ciphertext.

3.2 Incorporate Noise Filtering:  
To design a method that detects and temporarily removes non-alphabetic characters such as numbers, punctuation, and symbols (referred to as "noise") during the encryption and decryption processes. These characters will be stored with their positions and later reinserted to maintain the structure of the original text.

3.3 Prevent Brute Force Decryption:  
To expand the keyspace significantly by using a 3-letter key combination, resulting in 17,576 unique key possibilities (26 × 26 × 26), thereby making brute force attacks computationally expensive and highly impractical for unauthorized users.

3.4 Maintain Low Computational Complexity:  
To ensure that the algorithm remains simple, efficient, and suitable for environments with limited computing resources such as educational tools or lightweight applications, while still enhancing the level of encryption compared to the traditional Caesar Cipher.

3.5 Provide Complete Documentation and Examples:  
To produce comprehensive documentation that explains the logic, structure, and implementation of the algorithm, including step-by-step pseudocode, encryption and decryption examples, and analysis of security improvements. The documentation will serve both as a technical reference and an educational resource.

# 4. Overview of Caesar Cipher

The Caesar Cipher is one of the oldest and most well-known encryption techniques in the history of cryptography. Attributed to Julius Caesar, who reportedly used it to send secure messages to his military generals, the cipher operates by shifting each letter in the plaintext by a fixed number of positions down the alphabet. For example, with a shift of 3, the letter "A" becomes "D," "B" becomes "E," and so on. This process wraps around the alphabet, so that "Z" with a shift of 3 would become "C."

Mathematically, the Caesar Cipher can be represented using modular arithmetic. Each letter is first converted to a number from 0 to 25 (where A = 0, B = 1, ..., Z = 25), then a shift key k is added, and the result is taken modulo 26. That is:  
  
**Encrypted character = (Plaintext character + key) mod 26**  
**Decrypted character = (Ciphertext character - key + 26) mod 26**

This technique is categorized as a type of substitution cipher, where one character is replaced by another based on a rule. Its simplicity has made it a foundational concept in the study of cryptography, especially for beginners. However, its static nature means every character is shifted in the same way, which introduces predictable patterns that attackers can easily exploit.

Although effective in ancient times, the Caesar Cipher's key space is extremely limited. With only 25 possible shifts (excluding a shift of 0, which would leave the plaintext unchanged), it offers minimal protection against modern threats. An attacker could decrypt a Caesar-encrypted message by simply trying all possible shifts, a technique known as brute force. The use of frequency analysis — a technique that identifies the most common letters in a ciphertext — further reduces the effectiveness of this cipher.

Despite its limitations, the Caesar Cipher remains a fundamental concept in cryptographic education. It introduces key ideas such as substitution, key-based transformation, and modular arithmetic. Its limitations also serve as an educational tool, illustrating why modern encryption systems must incorporate larger keyspaces, randomness, and more complex transformations to ensure security.

In this project, we take inspiration from the Caesar Cipher’s foundational model and modify it with enhancements that significantly increase its security. By introducing a repeating key and ignoring non-letter characters during transformation, we create a cipher that preserves the simplicity of Caesar's method while eliminating many of its vulnerabilities.

# 5. Brute Force Vulnerabilities in Caesar Cipher

One of the most critical weaknesses of the classical Caesar Cipher is its vulnerability to brute force attacks. A brute force attack involves systematically trying every possible key until the correct one is found. In the case of the Caesar Cipher, the total number of possible shift values is only 25, since a shift of 0 would result in no encryption at all. This extremely limited keyspace makes brute force decryption not only feasible but trivially easy, especially with modern computing tools.

Unlike more complex encryption algorithms that rely on large keys or intricate transformations, the Caesar Cipher’s predictability simplifies the work of an attacker. Even without access to advanced software, a human can manually decrypt Caesar Cipher text within minutes by testing each possible shift. This simplicity, while advantageous for learning, renders the cipher virtually useless for securing confidential information in any real-world application.

Modern computers can break Caesar Cipher encryption in less than a second. A simple program can test all 25 possible keys and identify the correct plaintext by analyzing the resulting outputs. Some programs even use frequency analysis to identify the most likely decryption without testing every key. In English, certain letters like "E", "T", and "A" appear more frequently than others. If a ciphertext contains a high frequency of a particular letter, an attacker can guess that it might represent "E" and calculate the shift accordingly.

The Caesar Cipher also fails to incorporate any variability between characters. Every character is encrypted using the same shift, which means that repeated letters in the plaintext produce repeated letters in the ciphertext at equal intervals. This uniformity creates recognizable patterns that experienced cryptanalysts can exploit, making the encryption highly susceptible to statistical attacks in addition to brute force methods.

In summary, while the Caesar Cipher serves an important educational role in introducing the concept of encryption, its brute force vulnerabilities make it impractical for use in secure communications. This highlights the importance of expanding keyspace, introducing variability, and eliminating predictable patterns—all of which are addressed in the proposed enhanced algorithm introduced in this report.

# 6. Proposed Algorithm – Overview

The enhanced Caesar Cipher algorithm proposed in this report is designed to overcome the vulnerabilities of the traditional version, particularly its susceptibility to brute force attacks and inability to handle noise such as symbols, numbers, and punctuation. By introducing three key innovations—dynamic shifting using a repeating key, noise filtering, and character-by-character encryption—the algorithm significantly improves both the security and adaptability of the original cipher.

At the heart of the proposed method is the use of a fixed-length, **3-letter key**. Each character in the plaintext is encrypted by being combined with a corresponding character from this key. The key is repeated cyclically to match the length of the plaintext, creating a dynamic and variable shifting pattern. Instead of using a fixed numeric shift, each plaintext letter is shifted based on the alphabetical index (A = 0, B = 1, ..., Z = 25) of the current key character. For instance, if the key is "KEY", then the first plaintext letter is shifted by the position of 'K' (10), the second by 'E' (4), the third by 'Y' (24), then the cycle repeats for subsequent letters.

The second key innovation is **noise filtering**. During encryption, the algorithm identifies and temporarily removes non-alphabetic characters such as numbers, symbols, and punctuation marks. These characters are stored along with their original positions so that they can be accurately reinserted into the final encrypted message. This approach ensures that the encryption process applies only to meaningful alphabetic data while preserving the original formatting and structure of the input text.

This combination of repeated key-based shifting and selective character transformation results in a cipher that is highly resistant to brute force attacks. Instead of just 25 possible key values, the use of a 3-letter key increases the keyspace to **17,576 (26³)** unique combinations, making exhaustive search methods infeasible for most attackers. Additionally, the variable shifting disrupts recognizable patterns in the ciphertext, making frequency analysis much more difficult.

The proposed algorithm also maintains **low computational complexity**, making it ideal for lightweight applications, educational tools, and secure communication in constrained environments. It preserves the foundational logic of Caesar Cipher while enhancing it with modern cryptographic principles such as increased entropy, dynamic transformation, and structural integrity.

In the subsequent sections, we will present the detailed methodology for both encryption and decryption, describe the key repetition logic, and explain the noise handling mechanism. This will be followed by pseudocode, real-world examples, and a security analysis to validate the effectiveness of the proposed design.

# 7. Encryption Methodology

The encryption process in the proposed Caesar Cipher variant follows a structured and efficient procedure. It enhances the classic shift cipher by using a repeating three-letter key and ignoring non-alphabetic characters, which are later reinserted into the final encrypted message. This section outlines each step of the encryption algorithm, ensuring that every part of the process is clearly understood and logically sequenced.

## Step 1: Key Expansion and Preprocessing

The encryption process begins by accepting two inputs: the plaintext message and a three-letter key. The key is validated to ensure it contains only alphabetic characters and is exactly three characters long. If the key is shorter or longer, the system rejects the input to maintain algorithmic integrity. Once validated, the key is expanded to match the length of the plaintext (excluding noise characters) by repeating it cyclically. For example, if the key is "KEY" and the plaintext is "Information", the expanded key becomes "KEYKEYKEYKE".

## Step 2: Noise Character Identification

The plaintext is then scanned for non-alphabetic characters such as digits (0–9), punctuation marks (., ,, !, etc.), whitespace, and other symbols (@, #, $, etc.). These characters are classified as "noise" and temporarily removed from the text. Their original positions and values are stored in a list or map, allowing them to be restored accurately after the encryption process is complete. This step ensures that encryption is applied only to alphabetic characters, reducing complexity while preserving formatting.

## Step 3: Character-by-Character Encryption

After noise removal, the core encryption process begins. Each character in the cleaned plaintext is matched with a corresponding character from the expanded key. Both characters are converted into their alphabetical index values (A = 0, ..., Z = 25). The ciphered character is then calculated using the formula:

**EncryptedChar = (PlainCharIndex + KeyCharIndex) mod 26**

The resulting value is then converted back into a character using the inverse of the indexing process. For example, if the plaintext letter is 'I' (index 8) and the key letter is 'K' (index 10), the encrypted character is (8 + 10) mod 26 = 18, which corresponds to 'S'. This process continues for each alphabetic character in the message.

## Step 4: Reinsertion of Noise Characters

Once the encryption of all alphabetic characters is complete, the algorithm reinserts the stored noise characters into their original positions. This ensures that the final output closely resembles the structure of the original input, preserving spaces, numbers, symbols, and punctuation marks in their original order. For instance, if the plaintext was "Info123!mation", the digits 123 and the exclamation mark ! will be restored in the same positions in the ciphertext after the encrypted alphabetic characters are processed.

### Step 5: Final Output

The final result of the encryption process is a ciphertext string that maintains the structure of the original message while hiding the alphabetic content using a secure transformation based on a repeating key. This ciphertext can be safely transmitted or stored, with the assurance that it cannot be decrypted through brute force due to the expanded keyspace and variable shifting mechanism.

# 8. Decryption Methodology

The decryption process in the proposed Caesar Cipher variant is designed to accurately reverse the encryption while preserving the structure of the original message. By using the same 3-letter repeating key and stored noise positions, the algorithm ensures that each alphabetic character is decrypted to its original form, and all symbols, numbers, and formatting are restored without alteration. This section outlines the step-by-step decryption procedure.

## Step 1: Key Validation and Expansion

The decryption function begins with two inputs: the encrypted message (ciphertext) and the same 3-letter key used during encryption. The key must be validated to ensure it consists of exactly three alphabetic characters. Any deviation from this format results in an error. The key is then cyclically repeated to match the length of the alphabetic characters in the ciphertext (excluding noise), just as it was during encryption. For example, if the ciphertext contains 11 letters, the key "KEY" is expanded to "KEYKEYKEYKE".

## Step 2: Noise Character Extraction

The ciphertext is scanned to identify all non-alphabetic characters such as digits, whitespace, punctuation, and special symbols. These characters are extracted and their positions recorded using a map or list structure. This information is essential to reinsert the noise characters back into their original places after decrypting the alphabetic portion of the ciphertext.

## Step 3: Character-by-Character Decryption

After noise removal, the decryption of alphabetic characters begins. Each encrypted character is matched with the corresponding character from the expanded key. Both are converted to their alphabetical index values (A = 0, B = 1, ..., Z = 25). The original plaintext character is retrieved using the formula:

**PlainChar = (CipherCharIndex - KeyCharIndex + 26) mod 26**

This formula ensures the result is non-negative and within the range of 0 to 25. For example, if the encrypted character is 'S' (index 18) and the corresponding key character is 'K' (index 10), then the original character is (18 - 10 + 26) mod 26 = 34 mod 26 = 8, which corresponds to 'I'.

## Step 4: Reinsertion of Noise Characters

Once all alphabetic characters are decrypted, the algorithm reinserts the previously extracted noise characters into their recorded positions. This guarantees that the decrypted output mirrors the formatting of the original plaintext message, including the placement of numbers, punctuation, and special characters. For example, if the original plaintext was "Info123!mation", and the encrypted message was "Srgz123!wexscb", the decryption process will output "Info123!mation".

## Step 5: Final Decrypted Output

The final output of the decryption process is the original plaintext message with all structural and formatting elements restored. This method ensures that the encryption is fully reversible when the correct 3-letter key is provided, while making unauthorized decryption highly difficult due to the large keyspace and dynamic character mapping.

# 9. Noise Filtering Technique

The noise filtering technique is a fundamental component of the proposed Caesar Cipher variant. Unlike traditional ciphers that either ignore or inconsistently handle non-alphabetic characters, this algorithm introduces a structured approach to filtering and restoring "noise" — characters that do not fall within the range of uppercase or lowercase alphabetic letters. These may include digits, punctuation marks, whitespace, and special symbols. The filtering process ensures that only the meaningful content of the message is encrypted, while the overall structure and readability of the original text are preserved.

## Definition of Noise

For the purposes of this algorithm, noise is defined as any character that is not a letter from the English alphabet. This includes:

* Numeric digits (0–9)
* Whitespace (spaces, tabs, newlines)
* Punctuation marks (e.g., . , ? ! ; :)
* Symbols (e.g., @ # $ % ^ & \* ( ))
* Any other non-letter Unicode character

These characters are not encrypted or decrypted. Instead, they are temporarily excluded from the transformation process and reinserted later in their original positions.

## Filtering Process

During encryption and decryption, the algorithm first scans the input string character-by-character. For every character, it checks whether it is a letter (A–Z or a–z). If the character is not a letter, it is considered noise. The algorithm records the position (index) and value of each noise character in a list or dictionary structure. This mapping allows for precise restoration of the message's formatting after the encryption or decryption of alphabetic characters is complete.

For example, consider the plaintext:  
**"Hello, World! 123"**

The alphabetic characters extracted would be:  
**"HelloWorld"**  
The noise map would contain entries such as:

* Index 5 → ,
* Index 11 → !
* Index 12 → (space)
* Index 13 → 1
* Index 14 → 2
* Index 15 → 3

## Reinsertion Process

After the core transformation (encryption or decryption) of the alphabetic characters is complete, the algorithm reinserts each noise character back into its original position using the stored index map. This restoration process ensures that the final output closely mirrors the structure of the input message. The reinsertion is done after all alphabetic processing to maintain the correct indexing and avoid position shifts during transformation.

In the example above, if "HelloWorld" is encrypted to "RijvsUyvjn", reinserting the noise yields:  
**"Rijvs,Uyvjn! 123"**

## Advantages of Noise Filtering

The noise filtering technique offers several advantages:

* **Preserves formatting:** The final message retains its original layout, including spacing, punctuation, and symbols.
* **Reduces processing complexity:** Only alphabetic characters are processed, allowing for more efficient character mapping.
* **Enhances compatibility:** Messages containing structured text (e.g., email addresses, product codes) can be encrypted without corruption.
* **Improves usability:** End users can encrypt and decrypt naturally written sentences without worrying about special characters affecting results.

In conclusion, the noise filtering mechanism plays a vital role in maintaining message integrity while focusing encryption efforts on meaningful content. It bridges the gap between pure encryption logic and practical usability in real-world communication.

# 10. Key Repetition Mechanism

The core enhancement of the proposed Caesar Cipher variant lies in its use of a **repeating three-letter key**, which significantly increases security compared to the traditional fixed-shift Caesar Cipher. The key repetition mechanism dynamically cycles through the provided key, applying a different shift value to each letter of the plaintext in a repeating pattern. This section explains the logic, structure, and advantages of this mechanism in detail.

## Fixed-Length Key and Validation

The encryption and decryption processes require a key composed of exactly **three alphabetic characters**. This fixed-length key ensures simplicity in implementation while offering a dramatically larger keyspace than the original Caesar Cipher. With 26 letters in the English alphabet and three positions, the total number of possible keys is **26³ = 17,576**, making brute force attacks much more computationally demanding.

Before encryption begins, the key is validated to ensure it is alphabetic and exactly three characters long. If the key is shorter or longer than three characters, the system rejects it, maintaining algorithmic consistency and security expectations.

## Cyclical Key Expansion

Once validated, the three-letter key is repeated in a cyclical pattern to match the length of the input text (excluding noise characters). Each letter in the plaintext is paired with a corresponding letter from the repeated key. For example:

* **Plaintext (letters only):** Information
* **Key:** KEY
* **Expanded key:** KEYKEYKEYKE

The expansion is performed by iterating over the plaintext and, for each alphabetic character, assigning the next character from the key in a loop. Once the end of the key is reached, the cycle restarts from the first letter.

## Dynamic Shift Based on Key

Each key character is used to compute the shift value for the corresponding plaintext letter. The alphabetical position of the key character (A = 0, B = 1, ..., Z = 25) determines how many positions the plaintext letter is shifted during encryption or unshifted during decryption. This variability introduces unpredictability in the ciphertext, preventing the repetition of patterns that are typically present in traditional Caesar Cipher outputs.

For instance:

* I (8) + K (10) = S (18)
* n (13) + E (4) = R (17)
* f (5) + Y (24) = D (3) → since (5+24) mod 26 = 3

By cycling the key, the algorithm ensures that the same letter in different positions is encrypted differently, depending on its associated key character. This level of dynamism is a key contributor to the cipher's resistance against brute force and frequency analysis attacks.

## Synchronization Between Encryption and Decryption

It is essential that both encryption and decryption processes use the **same expanded key** based on the original 3-letter key and the alphabetic-only version of the message. This guarantees consistency in mapping and accurate reversal of the cipher. Any mismatch in key expansion or usage would result in incorrect decryption.

## Security Implications

The use of a repeating key achieves several important security outcomes:

* **Expanded keyspace** deters brute force decryption.
* **Position-dependent transformation** disrupts letter frequency patterns.
* **Symmetry of process** simplifies implementation while enhancing security.

In summary, the key repetition mechanism transforms the Caesar Cipher from a static, easily breakable cipher into a dynamic and position-sensitive encryption method. It forms the cryptographic backbone of the proposed algorithm, introducing necessary randomness and variability without increasing computational complexity.

# 11. Pseudocode for Encryption

This section outlines the pseudocode for the encryption process of the proposed Caesar Cipher variant. The algorithm uses a repeating three-letter key, filters out non-alphabetic characters (noise), and applies character-by-character shifting to produce a secure ciphertext. The pseudocode is written in a general, language-neutral format and can be adapted to any programming language.

### **Algorithm: EncryptCaesarVariant**

**Input:**

* plaintext ← original message (string)
* key ← 3-letter encryption key (string)

**Output:**

* ciphertext ← encrypted message with noise preserved

**Step 1: Initialize Structures**

1. cleanText ← empty string
2. noiseMap ← empty list of (index, character)
3. cipherTextChars ← empty list
4. expandedKey ← empty string
5. key ← convert to uppercase

**Step 2: Filter Noise and Extract Letters**  
6. For i from 0 to length(plaintext) - 1  
  If plaintext[i] is an alphabetic character:  
    Append plaintext[i] to cleanText  
  Else:  
    Append (i, plaintext[i]) to noiseMap

**Step 3: Expand the Key**  
7. For i from 0 to length(cleanText) - 1  
  Append key[i mod 3] to expandedKey

**Step 4: Encrypt Each Letter**  
8. For i from 0 to length(cleanText) - 1  
  plainChar ← uppercase of cleanText[i]  
  keyChar ← uppercase of expandedKey[i]  
  pIndex ← ASCII value of plainChar - ASCII of 'A'  
  kIndex ← ASCII value of keyChar - ASCII of 'A'  
  cIndex ← (pIndex + kIndex) mod 26  
  cipherChar ← character with ASCII value (cIndex + ASCII of 'A')  
  Append cipherChar to cipherTextChars

**Step 5: Reinsert Noise Characters**  
9. For each (position, char) in noiseMap:  
  Insert char into cipherTextChars at index position

**Step 6: Return Final Ciphertext**  
10. Return join(cipherTextChars) as the final encrypted message

### **Example Use Case:**

Plaintext: "Info123!mation"

Key: "KEY"

CleanText: "Information"

ExpandedKey: "KEYKEYKEYKE"

Encrypted: "Srgz123!wexscb"

This pseudocode enables anyone to implement the encryption algorithm efficiently and securely, with clarity in handling both text and noise. The structured nature ensures it can be translated into Python, Java, JavaScript, C#, or any other language with basic string manipulation support.

# 12. Pseudocode for Decryption

The decryption process reverses the encryption logic by using the same three-letter key and restoring the original plaintext. The algorithm extracts only the alphabetic characters, uses the key to apply reverse shifts, and then re-inserts the non-alphabetic noise characters into their original positions. This ensures accurate recovery of the original message structure.

### **Algorithm: DecryptCaesarVariant**

**Input:**

* ciphertext ← encrypted message (string)
* key ← 3-letter decryption key (string)

**Output:**

* plaintext ← original message with noise restored

**Step 1: Initialize Structures**

1. cleanCipher ← empty string
2. noiseMap ← empty list of (index, character)
3. plainTextChars ← empty list
4. expandedKey ← empty string
5. key ← convert to uppercase

**Step 2: Filter Noise and Extract Letters**  
6. For i from 0 to length(ciphertext) - 1  
  If ciphertext[i] is an alphabetic character:  
    Append ciphertext[i] to cleanCipher  
  Else:  
    Append (i, ciphertext[i]) to noiseMap

**Step 3: Expand the Key**  
7. For i from 0 to length(cleanCipher) - 1  
  Append key[i mod 3] to expandedKey

**Step 4: Decrypt Each Letter**  
8. For i from 0 to length(cleanCipher) - 1  
  cipherChar ← uppercase of cleanCipher[i]  
  keyChar ← uppercase of expandedKey[i]  
  cIndex ← ASCII value of cipherChar - ASCII of 'A'  
  kIndex ← ASCII value of keyChar - ASCII of 'A'  
  pIndex ← (cIndex - kIndex + 26) mod 26  
  plainChar ← character with ASCII value (pIndex + ASCII of 'A')  
  Append plainChar to plainTextChars

**Step 5: Reinsert Noise Characters**  
9. For each (position, char) in noiseMap:  
  Insert char into plainTextChars at index position

**Step 6: Return Final Plaintext**  
10. Return join(plainTextChars) as the final decrypted message

### **Example Use Case:**

Ciphertext: "Srgz123!wexscb"

Key: "KEY"

CleanCipher: "Srgzwexscb"

ExpandedKey: "KEYKEYKEYKE"

Decrypted: "Info123!mation"

This pseudocode ensures a clean and accurate decryption process when the correct 3-letter key is provided. It mirrors the encryption logic, allowing seamless and reliable restoration of the original message, even when noise characters are present.

# 13. Implementation Example

To demonstrate the practical functionality of the proposed Caesar Cipher variant, this section provides a detailed walkthrough of the encryption and decryption processes using a sample plaintext message and a three-letter key. This real-world example showcases the step-by-step transformation of the data, reinforcing the theoretical concepts outlined in earlier sections.

### **Example Parameters**

* **Plaintext:** Information is Power! 2025
* **Key:** KEY

### **Step 1: Identify Alphabetic and Noise Characters**

**Alphabetic portion:**  
InformationisPower  
(18 alphabetic characters)

**Noise characters with positions:**

* Position 11: (space)
* Position 12: i → alphabetic
* Position 13: s → alphabetic
* Position 14: (space)
* Position 15: P → alphabetic
* Position 16: o → alphabetic
* Position 17: w → alphabetic
* Position 18: e → alphabetic
* Position 19: r → alphabetic
* Position 20: !
* Position 21:
* Position 22: 2
* Position 23: 0
* Position 24: 2
* Position 25: 5

**Cleaned alphabetic string:**  
InformationisPower

**Expanded Key:**  
KEYKEYKEYKEYKEYKEYKE  
(length = 18 to match number of letters)

### **Step 2: Encrypt the Message**

We apply the encryption formula for each letter:

**Encrypted Characters (first few):**

* I (8) + K (10) → S (18)
* n (13) + E (4) → R (17)
* f (5) + Y (24) → D (3)
* o (14) + K (10) → Y (24)
* r (17) + E (4) → V (21)
* m (12) + Y (24) → K (10)
* a (0) + K (10) → K (10)
* t (19) + E (4) → X (23)
* i (8) + Y (24) → G (6)
* o (14) + K (10) → Y (24)
* n (13) + E (4) → R (17)
* i (8) + Y (24) → G (6)
* s (18) + K (10) → C (2)
* P (15) + E (4) → T (19)
* o (14) + Y (24) → M (12)
* w (22) + K (10) → G (6)
* e (4) + E (4) → I (8)
* r (17) + Y (24) → P (15)

**Encrypted Text (without noise):**  
SRDYVKKXGYRGCTMGIP

### **Step 3: Reinsert Noise**

Original noise:

* Position 11 →
* Position 20 → !
* Position 21 →
* Position 22 → 2
* Position 23 → 0
* Position 24 → 2
* Position 25 → 5

**Final Ciphertext:**  
SRDYVKKXGYR GC!TMGIP 2025

### **Step 4: Decrypting the Ciphertext**

Following the decryption logic with the same key (KEY), the encrypted message SRDYVKKXGYR GC!TMGIP 2025 is processed:

1. Remove noise → extract SRDYVKKXGYRGCTMGIP
2. Apply reverse shifting using the key KEYKEYKEYKEYKEYKEYKE
3. Result → INFORMATIONISPOWER
4. Reinsert original noise positions →  
   **Final decrypted message:**  
   Information is Power! 2025

This example illustrates the algorithm’s ability to:

* Accurately encrypt and decrypt alphabetic content
* Preserve original formatting and symbols
* Prevent brute force decryption by introducing variable shifts via a repeating key

# 14. Security Analysis

The security of any cryptographic algorithm lies in its ability to resist unauthorized decryption through various attack vectors, including brute force, frequency analysis, and pattern recognition. The enhanced Caesar Cipher variant presented in this report addresses the inherent vulnerabilities of the classical Caesar Cipher and significantly improves its resilience against such attacks through several strategic modifications.

## 1. **Expanded Keyspace and Brute Force Resistance**

The traditional Caesar Cipher is particularly vulnerable due to its limited keyspace of only 25 possible shifts. In contrast, the proposed algorithm employs a 3-letter key, each character chosen from the 26-letter English alphabet. This expands the total number of unique keys to 26³ = **17,576 possible combinations**, rendering brute force attacks computationally impractical for casual attackers, especially when the ciphertext contains minimal contextual clues.

Furthermore, since the key shifts vary at each character position based on the key’s repeated sequence, brute force is no longer as simple as trying 17,576 full decryptions. Each incorrect key produces drastically different outputs, making human pattern recognition or automated scanning far less effective.

## 2. **Disruption of Frequency Patterns**

The classic Caesar Cipher applies a uniform shift to every character, allowing attackers to use **frequency analysis**—a technique that identifies high-frequency letters like ‘E’, ‘T’, and ‘A’ in English—to infer the key. The proposed variant circumvents this issue by assigning **different shifts per character**, based on the cyclical key. As a result, even repeated letters in the plaintext are encrypted differently depending on their position and corresponding key character.

This variability destroys predictable frequency distributions in the ciphertext, severely weakening frequency-based decryption strategies and statistical attacks.

## 3. **Noise Filtering and Structural Preservation**

The encryption algorithm temporarily removes non-letter characters (digits, punctuation, and symbols), ensuring that these elements do not interfere with the encryption logic. However, their positions and values are recorded and accurately reinserted into the ciphertext or plaintext. This **noise filtering** not only improves processing efficiency but also enhances the **confusion** property of the cipher by preserving the structure of the original message while altering the underlying data.

By retaining the natural spacing and formatting of the message, the encrypted text remains readable in form but incomprehensible in content—an effective balance between security and usability.

## 4. **Symmetric Key Dependence**

The system relies on a **symmetric key model**, meaning the same key is required for both encryption and decryption. This ensures that even if an encrypted message is intercepted, it cannot be decrypted without possession of the correct 3-letter key. Since the key is fixed in size and not transmitted with the message, the **security relies entirely on keeping the key confidential**, a fundamental principle in symmetric cryptography.

## 5. **Limitations and Attack Considerations**

While the proposed cipher significantly improves upon the traditional Caesar Cipher, it is not immune to all forms of cryptographic attack. For example:

* If an attacker has **prior knowledge of the algorithm and a sample plaintext-ciphertext pair**, a known-plaintext attack could theoretically reveal the key.
* The use of only 3 characters for the key, though much more secure than a single shift, is still relatively small compared to modern encryption standards like AES or RSA.

Nevertheless, for lightweight applications, educational tools, or controlled communication systems where simplicity, reversibility, and low computational cost are essential, this algorithm provides a **strong level of security** relative to its complexity.