International Journal of Multidisciplinary Research and Growth Evaluation.

Build A Practice Exercise on Secure Encryption and Decryption using A Character Substitution Algorithm

Vuong Thuy Linh [1*], Dang Phan Thu Huong [2]
[1] Master, Lecturer, Faculty of Information Technology, University of Labour and Social Affairs, Vietnam
[2] Faculty of Information Technology, University of Labour and Social Affairs, No. 43, Tran Duy Hung Street, TrungHoa Ward, CauGiay District, Hanoi City, Vietnam

* Corresponding Author: **Vuong Thuy Linh**

Article Info
ISSN (online): 2582-7138
Volume: 06
Issue: 02
March-April 2025
Received: 20-02-2025
Accepted: 19-03-2025
Page No: 1290-1297

Abstract
In the digital era, where data is a vital asset, ensuring information security has become increasingly critical, especially in contexts involving confidential communication such as operational instructions or personal data exchange. Among various cryptographic techniques, substitution ciphers represent one of the fundamental methods of encryption and decryption, offering an intuitive approach to concealing information by systematically replacing characters according to a predefined scheme. Although considered relatively simple compared to modern cryptographic standards, substitution ciphers remain an effective pedagogical tool for introducing core concepts in information security and algorithm design.
This paper introduces the design and implementation of a practical exercise that enables students to explore secure encryption and decryption using a character substitution algorithm. The system operates based on a substitution table where each character in the plaintext is replaced by a corresponding character in a secret key mapping. This approach simulates a classical symmetric encryption scheme and allows learners to directly engage with the encryption process, understand the mechanics of cipher construction, and reflect on the importance of key secrecy and structural design.
The main objectives of this exercise are threefold: (1) to provide a hands-on experience that reinforces theoretical knowledge of substitution encryption; (2) to facilitate the development of algorithmic thinking and programming skills through the design and implementation of encryption-decryption functions; and (3) to foster critical evaluation of the security level offered by character substitution methods in different practical contexts. By working through the exercise, students gain exposure to the trade-offs between simplicity and security, recognize potential vulnerabilities such as frequency analysis, and consider how modern encryption schemes address these limitations.
The exercise was deployed in an undergraduate-level information security course, and experimental results based on student feedback and assessment performance showed promising outcomes. Students demonstrated improved understanding of cryptographic principles, greater engagement with algorithm development, and increased awareness of real-world challenges in information protection. Furthermore, the interactive nature of the exercise encouraged collaborative problem-solving and stimulated discussions on the evolution of encryption techniques.
In summary, this work highlights the educational value of integrating practical cryptographic exercises into the curriculum. The character substitution-based encryption-decryption activity serves not only as a foundation for learning basic security mechanisms but also as a platform for promoting analytical thinking, code implementation skills, and a deeper appreciation for the complexities of securing digital communication. Future work may include extending this exercise with more advanced cipher variants or integrating cryptographic attack simulations to further enhance student comprehension and readiness for cybersecurity challenges.

Keywords: Encryption, Decryption, Transposition, Information Security

1. Introduction

In the context of increasing threats to information security, data protection has become an urgent requirement across all fields. Among the various methods for ensuring information safety, data encryption remains one of the most effective solutions, allowing sensitive content to be protected from unauthorized access during transmission or storage.
Throughout the history of cryptographic development, the character substitution algorithm has played a significant role in both education and research due to its simplicity, accessibility, and its ability to clearly illustrate fundamental concepts of symmetric encryption [4].

This algorithm operates on the principle of replacing each character in the plaintext with another character based on a predefined rule. Prominent variations of the substitution cipher include the Caesar cipher, Vigenère cipher, and multi-alphabet encryption systems [3].

Despite its straightforward structure, substitution encryption continues to find practical applications in various domains such as personal data protection, secure communications, and identification and access control systems [1, 5]. In scenarios requiring a high level of reliability, such as preventing adversaries from deciphering critical messages, the substitution cipher can still prove effective in simple but security-critical situations.

Recent studies have proposed several enhancements to improve the security of substitution encryption algorithms. For instance, integrating substitution ciphers with chaotic encryption methods can significantly increase the complexity of cryptanalysis, thereby enhancing resistance to decryption attacks [1]. Other research efforts have introduced the use of three-step Vigenère tables to better visualize the encryption process [3]. In addition, evaluating and comparing the security levels of different variations—such as Caesar cipher, columnar transposition, and row transposition—has provided a foundation for selecting the most appropriate solution for specific applications [2].

In parallel with algorithm development, cryptanalysis techniques also play a vital role in assessing the security of encryption systems. Advances in artificial intelligence and evolutionary computation, such as swarm intelligence optimization, cuckoo search, and genetic algorithms, have been successfully applied to break substitution cipher systems, thereby helping to identify weaknesses and improve algorithm robustness [6, 7].

Based on these practical needs and academic values, this paper proposes the development of a simulation program for encryption and decryption using a character substitution algorithm, with a predefined secret key implemented in a substitution table. The specific objective of the program is to secure a sensitive operational message: "NO. 43, TRAN DUY HUNG STREET, TRUNG HOA WARD, CAU GIAY DISTRICT, HANOI CITY, VIETNAM", by encrypting and then decrypting this content using a defined substitution rule. The development of this simulation program not only serves to validate the practical applicability of the substitution cipher but also provides a valuable teaching tool for information security training. The following sections of the paper will present a detailed explanation of the algorithm, the implementation of the simulation program, and an evaluation of the security level offered by this method.

2. Problem Statement

The objective of this problem is to develop a simulation program for encryption and decryption using a character substitution cipher method. This method relies on a predefined substitution table that serves as a secret key. The program will perform the following tasks:

Encryption: Transform the original message into a ciphertext by substituting each character based on the substitution table.

Decryption: Restore the original message from the ciphertext using the same substitution table.

The sensitive information to be protected is a specific address:

“NO. 43, TRAN DUY HUNG STREET, TRUNG HOA WARD, CAU GIAY DISTRICT, HANOI CITY, VIETNAM”

The substitution table will be defined in advance, for example, as a one-to-one mapping between characters in the standard alphabet and a shuffled or rule-based character set.

The program must ensure a one-to-one correspondence in encryption and decryption, preserve data integrity, and allow for extension to various character sequences.

Substitution Table:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
X	M	O	P	T	L	K	Q	N	U	Y	S	E	V	R	Z	D	C	W	B	G	H	A	J	F	I

3. Objectives

This practical exercise helps students’ approach and apply the character substitution cipher algorithm in information encryption and decryption. Upon completion of the exercise, students will achieve the following objectives:

Understand the principle of character substitution encryption: Gain a thorough understanding of how the character substitution algorithm works, including how to map original characters to substituted characters based on a fixed encoding table, and how to decrypt by reverse lookup of the table.

Program the encryption and decryption algorithm: Write MATLAB source code to perform the encryption and decryption of text using the character substitution method, ensuring accuracy and efficiency.

Develop a simple user interface: Design an interface for entering text, performing encryption/decryption, and displaying the results visually for ease of use during the practical session and testing of the algorithm.

Analyze and evaluate the results: Compare the original text, encrypted text, and decrypted text to verify the accuracy of the algorithm. Assess the security level of the encryption method by analyzing the possibility of decryption without the substitution table.

Raise awareness of data security: Understand the real-world applications of character substitution encryption, particularly in protecting information in military, communication security, and personal data protection sectors.

Through this practical exercise, students will not only reinforce theoretical knowledge of encryption but also develop programming skills, data processing abilities, and evaluate the security of an information protection method.

4. Tools and software used

To carry out the character substitution encryption and decryption exercise, students are required to use the following tools and software:

Matlab: A powerful programming and simulation software that supports matrix processing, string manipulation, and user interface (GUI) programming. Matlab will be used to implement the character substitution encryption and decryption algorithm, as well as to build the user interface and perform string manipulation operations.

Matlab Editor: An integrated development environment (IDE) for writing, editing, and running the source code for the encryption and decryption algorithm. Matlab Editor provides powerful tools for developing and testing the source code, ensuring the program functions correctly.

Matlab String Processing Library: Includes functions such as

reshape, mod, char, and double for manipulating text data, facilitating efficient encryption and decryption operations. These functions will help convert characters to numerical codes and vice versa, as well as reorganize data into appropriate structures.

Personal Computer: A computer with Matlab installed, configured to run encryption and decryption programs efficiently. The computer should have sufficient resources to perform operations and simulate the algorithm without encountering performance issues.

Guidelines on Character Substitution Encryption Algorithm: Documents, papers, and academic resources will assist students in gaining a deeper understanding of how the character substitution encryption algorithm works, the steps involved, and how to apply this method in practice.

Students should ensure their working environment is set up with Matlab and that they are familiar with the basic operations of the software before starting the practical exercise. This ensures a smooth practice session and enables students to effectively apply theoretical knowledge in a practical setting.

5. Implementation Procedure

To effectively construct a simulation program for character substitution encryption and decryption using a predefined secret key in the form of a substitution table, the implementation process should follow a structured, step-by-step methodology. This approach ensures both functionality and pedagogical value, allowing learners to comprehend not only the technical process but also the underlying principles of classical cryptography.

Step 1: Define the Substitution Table

The initial step involves the definition of a substitution table, which serves as the secret key for both encryption and decryption. This table consists of a one-to-one mapping between the standard alphabet (typically uppercase letters A–Z) and a permuted version of the same set.

Substitution Table Description: Each character in the standard alphabet is assigned a corresponding character from a rearranged alphabet. This mapping will dictate how each plaintext character is converted during encryption.

Secret Key: The secret key, in this context, is a string representing the rearranged alphabet. It governs the substitution rules and must remain confidential to ensure data security.

Step 2: Preprocessing the Input Text

Prior to encryption, the input text undergoes a preprocessing phase to simplify the encoding process while preserving the structure of the original message.

Whitespace Removal: All whitespace characters (e.g., spaces, tabs) and potentially non-alphabetic symbols are removed from the plaintext to prepare a clean string for encryption.

Whitespace Position Logging: To maintain reversibility and ensure accurate reconstruction during decryption, the positions of all removed whitespace characters are recorded and stored. This metadata will be re-applied during the post-decryption phase to restore the original formatting.

Step 3: Encryption Process

Once the input text has been preprocessed, the encryption algorithm is applied.

Character Substitution: Each character in the processed plaintext is replaced by its corresponding character in the substitution table. This operation is performed sequentially across the entire input string, generating the encrypted output (ciphertext).

Step 4: Decryption Process

Decryption is performed by reversing the character substitution process using an inverse mapping derived from the original substitution table.

Reverse Mapping: A reverse substitution table is constructed to restore each character in the ciphertext to its original form in the standard alphabet.

Reinsertion of Whitespace: Using the positions logged during the preprocessing stage, the program re-inserts whitespace characters into the appropriate locations in the decrypted text. This step ensures that the final output accurately reflects the structure of the original input.

Step 5: User Interface Development

To enhance usability and provide an interactive learning experience, a graphical user interface (GUI) is integrated into the simulation program.

Input Mechanism: The GUI should support input of both plaintext and ciphertext, depending on the desired operation (encryption or decryption).

Functionality Controls: Buttons or selection options should enable users to choose between encryption and decryption, triggering the appropriate function within the application.

Output Display: The results of the encryption or decryption process must be clearly displayed in a designated output field, allowing users to easily observe and verify the transformations.

Step 6: Testing and Evaluation

The final phase involves systematic testing of the developed application to ensure accuracy, functionality, and security.

Functional Validation: The correctness of the encryption-decryption process is verified by comparing the decrypted output with the original input text. A match indicates successful implementation of the algorithm.

Security Assessment: A preliminary analysis of the substitution cipher's ability to protect sensitive data is conducted. While simple substitution is vulnerable to frequency analysis, it remains useful in educational and low-risk contexts. The analysis should address the cipher’s strengths and limitations, particularly its susceptibility to cryptanalytic attacks.

By following this implementation procedure, students not only acquire hands-on experience in developing a classical cryptographic system but also gain essential knowledge in data preprocessing, reversible transformation, interface design, and security evaluation. This holistic approach serves both educational and practical objectives in the field of information security.

6. Code Implementation

```
function SUBSTITUTION_CIPHER
    % Create the main interface
    fig = figure ('Name', 'SUBSTITUTION CIPHER', 'Position', [500 200 650 450]);
    % Standard size
    labelWidth = 200;
    boxWidth = 550;
    boxHeight = 35;
    spacing = 20;  % Spacing between elements
    startY = 360;  % Starting position of the first box
    % Substitution table input box
    uicontrol('Style', 'text', 'Position', [20, startY, labelWidth, boxHeight], 'String', 'SUBSTITUTION TABLE:', ...
            'FontWeight', 'bold', 'FontSize', 12, 'HorizontalAlignment', 'left');
    customTableBox = uicontrol('Style', 'edit', 'Position', [210, startY+5, boxWidth, boxHeight-5], 'String', '', 'FontWeight', 'bold');
    % Original text input box
    uicontrol('Style', 'text', 'Position', [20, startY - (boxHeight + spacing), labelWidth, boxHeight], 'String', 'ORIGINAL TEXT:', ...
            'FontWeight', 'bold', 'FontSize', 12, 'HorizontalAlignment', 'left');
    inputBox = uicontrol('Style', 'edit', 'Position', [210, startY - (boxHeight + spacing), boxWidth, boxHeight], 'String', '', 'FontWeight', 'bold');
    % Encrypted text display box
    uicontrol('Style', 'text', 'Position', [20, startY - 2*(boxHeight + spacing), labelWidth, boxHeight], 'String', 'ENCRYPTED TEXT:', ...
            'FontWeight', 'bold', 'FontSize', 12, 'HorizontalAlignment', 'left');
    encodedBox = uicontrol('Style', 'edit', 'Position', [210, startY - 2*(boxHeight + spacing), boxWidth, boxHeight], 'String', '', ...
                'Enable', 'inactive', 'FontWeight', 'bold');
    % Decrypted text display box
    uicontrol('Style', 'text', 'Position', [20, startY - 3*(boxHeight + spacing), labelWidth, boxHeight], 'String', 'DECRYPTED TEXT:', ...
            'FontWeight', 'bold', 'FontSize', 12, 'HorizontalAlignment', 'left');
    decodedBox = uicontrol('Style', 'edit', 'Position', [210, startY - 3*(boxHeight + spacing), boxWidth, boxHeight], 'String', '', ...
                'Enable', 'inactive', 'FontWeight', 'bold');
    % Buttons
    buttonWidth = 120;
    buttonHeight = 45;
    buttonY = startY - 4*(boxHeight + spacing) - 30;
    uicontrol('Style', 'pushbutton', 'Position', [210, buttonY, buttonWidth, buttonHeight], 'String', 'ENCRYPT', ...
            'FontWeight', 'bold', 'FontSize', 12, 'ForegroundColor', 'k', 'Callback', @encrypt_callback);
    uicontrol('Style', 'pushbutton', 'Position', [350, buttonY, buttonWidth, buttonHeight], 'String', 'DECRYPT', ...
            'FontWeight', 'bold', 'FontSize', 12, 'ForegroundColor', 'k', 'Callback', @decrypt_callback);
    uicontrol('Style', 'pushbutton', 'Position', [500, buttonY, buttonWidth, buttonHeight], 'String', 'CLEAR ALL', ...
            'FontWeight', 'bold', 'FontSize', 12, 'ForegroundColor', 'k', 'Callback', @clear_callback);
    % Default substitution table (Caesar shift 3)
    defaultSubstitutionTable = 'DEFGHIJKLMNOPQRSTUVWXYZABC';
    % Encryption function
    % Encryption function
    function encrypt_callback(~, ~)
        plainText = upper(get(inputBox, 'String')); % Convert to uppercase
        customTable = get(customTableBox, 'String'); % Get the substitution table from the user
        if isempty(customTable)
            substitutionTable = defaultSubstitutionTable;
        else
            if length(customTable) ~= 26
                errordlg('The substitution table must contain exactly 26 letters!', 'Error');
                return;
            end
            substitutionTable = upper(customTable);
        end
        [cipherText, spacePositions] = substitution_encrypt(plainText, substitutionTable);
        set(encodedBox, 'String', cipherText);
        set(decodedBox, 'UserData', spacePositions); % Store space positions
    end
    % Decryption function
    function decrypt_callback(~, ~)
        cipherText = get(encodedBox, 'String');
        customTable = get(customTableBox, 'String');
        if isempty(customTable)
            substitutionTable = defaultSubstitutionTable;
        else
            if length(customTable) ~= 26
                errordlg('The substitution table must contain exactly 26 letters!', 'Error');
                return;
            end
            substitutionTable = upper(customTable);
        end
        spacePositions = get(decodedBox, 'UserData'); % Get stored space positions
        decryptedText = substitution_decrypt(cipherText, substitutionTable, spacePositions);
        set(decodedBox, 'String', decryptedText);
    end
    % Clear all function
    function clear_callback(~, ~)
        set(inputBox, 'String', '');
        set(encodedBox, 'String', '');
        set(decodedBox, 'String', '');
        set(customTableBox, 'String', '');
```

```
    end
    % Substitution encryption function
    function [cipherText, spacePositions] =
substitution_encrypt(plainText, substitutionTable)
        % Initialize the variable for encrypted text
        cipherText = '';
        spacePositions = []; % Store space positions
        % Loop through each character of the plain
text
        for i = 1:length(plainText)
            char = plainText(i);
            % If it's a space, save its position and skip
            if char == ' '
                cipherText = [cipherText, ' '];
                spacePositions = [spacePositions, i];
            elseif isletter(char)
                % Encrypt the character using the
substitution table
                charIndex = double(upper(char)) -
double('A') + 1; % Get the index of the character
(A=1, B=2, ...)
                cipherChar =
substitutionTable(charIndex); % Encrypt the
character
                cipherText = [cipherText, cipherChar];
            else
                % If the character is not a letter, keep it
 unchanged
                cipherText = [cipherText, char];
            end
        end
    end
    % Substitution decryption function
    function decryptedText =
 substitution_decrypt(cipherText, substitutionTable,
 spacePositions)
        % Initialize the variable for decrypted text
        decryptedText = '';
        reverseSubstitutionTable =
 reverse_substitution_table(substitutionTable); %
 Create the reverse substitution table
        % Loop through each character of the cipher
 text
        for i = 1:length(cipherText)
            char = cipherText(i);
            % If it's a space, keep it unchanged
            if char == ' '
                decryptedText = [decryptedText, ' '];
            elseif isletter(char)
                % Decrypt the character using the
 reverse substitution table
                charIndex = find(substitutionTable ==
 char); % Get the index of the character in the
 substitution table
                decryptedChar =
 reverseSubstitutionTable(charIndex); % Decrypt
 the character
                decryptedText = [decryptedText,
 decryptedChar];
            else
                % If the character is not a letter, keep it
 unchanged
                decryptedText = [decryptedText, char];
            end
        end
    end
    % Function to create the reverse substitution
 table
    function reverseTable =
 reverse_substitution_table(substitutionTable)
        % Create the reverse substitution table
 (converts from cipher to original letter)
        reverseTable = '';
        for i = 1:length(substitutionTable)
            reverseTable = [reverseTable, char('A' + i -
 1)];
        end
    end
end
```

7. Results and Evaluation

7.1. Experimental Results

The encryption and decryption system using the character substitution algorithm has been designed to allow text encryption and decryption through the use of a substitution table. The substitution table can either be a default table (Caesar Shift 3) or a custom table entered by the user. The system interface provides tools for users to input text, encrypt, decrypt, and clear data in a user-friendly manner.

- The main interface includes input fields and function buttons to assist users in the process of encrypting and decrypting text:
  Substitution table input field: A substitution table can be entered here. If no table is entered, the system defaults to using the Caesar Shift table with a shift of three characters (e.g., A → D, B → E, C).
  Original text input field: The text to be encrypted is entered here. The system supports the encryption of any text, including letters, punctuation, and whitespace.
  Encrypted text display field: After encrypting the text, the result is displayed in this field. The text will be replaced by the corresponding characters from the substitution table.
  Decrypted text display field: After decrypting the encrypted text, the result is displayed in this field. The result will correspond to the original text if the decryption process is performed correctly.
- The main functions of the system are organized through three function buttons:
  Encrypt: When the ENCRYPT button is pressed, the original text is encrypted according to the substitution table, and the result is displayed in the "ENCRYPTED TEXT" field.
  Decrypt: The DECRYPT button allows the user to decrypt text that was previously encrypted. The decryption result will be displayed in the "DECRYPTED TEXT" field.
  Clear all: The CLEAR ALL button allows the user to clear all input fields and current results, facilitating a fresh start for encryption and decryption with new data.
- Encryption and Decryption Process:
  Encryption: The original text is entered into the ORIGINAL TEXT field. The substitution table is selected from the SUBSTITUTION TABLE field, or the system will use the default table if this field is empty. When the ENCRYPT button is pressed, the original text is encrypted and the result is displayed in the ENCRYPTED TEXT field.
  Decryption: The encrypted text from the ENCRYPTED TEXT field is decrypted when the DECRYPT button is pressed. The decryption process uses the entered substitution table, and the result is displayed in the

DECRYPTED TEXT field.
Clear data: The CLEAR ALL button will erase all input fields and results, allowing the user to start with new data.

- Notes on using the system:
  Substitution table: Ensure that the custom substitution table contains all 26 letters. If not, the system will prompt an error and request the user to enter a valid substitution table.
  Non-alphabetic characters: Non-alphabetic characters (e.g., punctuation, numbers, etc.) will not be encrypted and will remain unchanged during encryption and decryption.
  Whitespace: Whitespace in the original text will be preserved during both encryption and decryption.

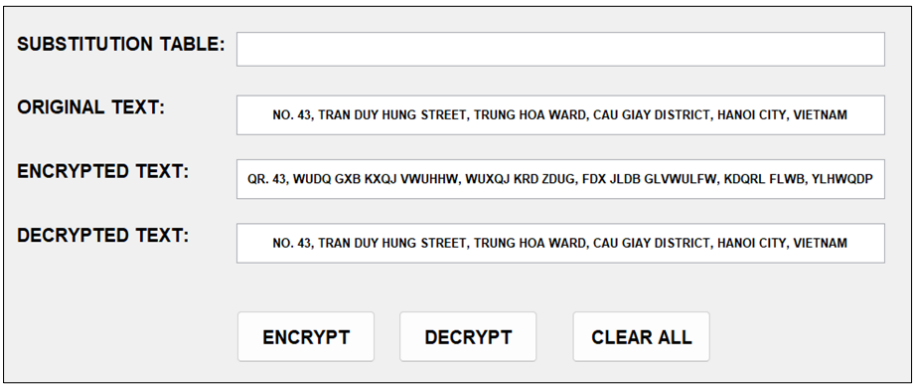
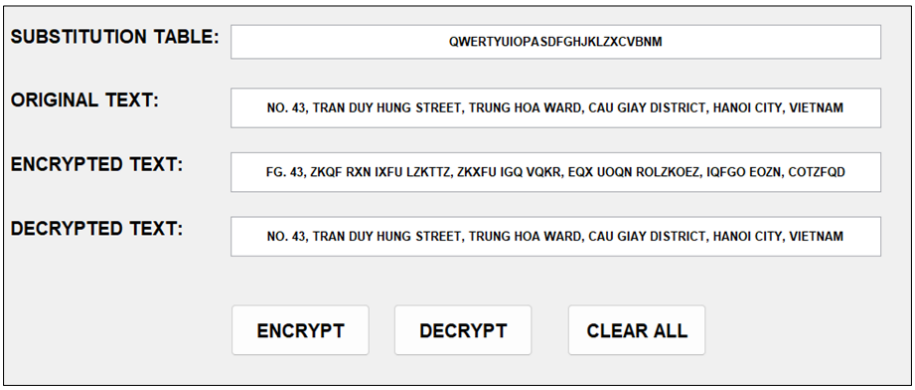


Fig 1: Results of Encoding and Decoding Using the Default Caesar Shift Character Substitution Table



(e.g., QWERTYUIOPASDFGHJKLZXCVBNM)

Fig 2: Results of Encoding and Decoding Using the Custom 26-Letter Character Substitution Table

7.2 Evaluation of Effectiveness

The implementation and testing of the developed program, which employs the Substitution Cipher algorithm for both encryption and decryption of textual data, demonstrate notable results in terms of accuracy, usability, security, and performance. This section provides a comprehensive assessment of the algorithm's effectiveness based on experimental trials and theoretical analysis.

- Security Evaluation: From a security standpoint, the strength of the substitution cipher is highly dependent on the type of substitution table used. When utilizing a Caesar Shift cipher — a specific case of the substitution cipher where each letter is shifted by a fixed number of positions — the encryption exhibits a low level of security. Due to its predictable structure and fixed displacement, the Caesar cipher is particularly vulnerable to frequency analysis and other classical cryptanalytic techniques. The limited key space (only 25 possible shifts) makes brute-force attacks trivial, rendering the method unsuitable for scenarios requiring robust data protection.
- In contrast, employing a randomly generated substitution table consisting of a full permutation of the 26 letters significantly enhances the cipher's resistance to attacks. The number of possible permutations (26!) creates a substantially larger key space, making brute-force attacks computationally infeasible for casual adversaries. While still susceptible to frequency-based analysis due to the preservation of character frequency patterns, the randomized substitution table offers a considerably higher level of confidentiality compared to the Caesar cipher.
- Accuracy and Functional Validation: The program ensures high fidelity in the encryption-decryption cycle. The decrypted output is an exact replica of the original input text, with no character distortion, data loss, or misplacement of symbols. This precision is achieved through the meticulous handling of text preprocessing steps (e.g., whitespace removal and position logging) and the systematic reapplication of these structural elements after decryption. Extensive experimental trials confirm that the algorithm operates deterministically and consistently across various input texts, thereby validating the correctness and stability of the implementation.
- Automation and Usability: One of the strengths of the developed software lies in its high degree of automation and user-friendliness. The graphical user interface

allows users to seamlessly enter plaintext or ciphertext, select the desired operation (encryption or decryption), and instantly view the results with a single button press. The inclusion of additional features, such as a "Clear" function to reset inputs and outputs, enhances usability by facilitating multiple rounds of testing without restarting the application. The intuitive design ensures accessibility even for users with limited knowledge of cryptographic algorithms.

- Practical Applicability: While the substitution cipher — especially when enhanced with a random substitution table — can be applied in scenarios where lightweight or internal data protection is sufficient (e.g., basic educational tools, internal messaging in low-risk systems), it falls short of providing adequate protection for sensitive or high-value information. The deterministic nature of the cipher and the preservation of character frequencies expose it to known-plaintext and ciphertext-only attacks in more advanced threat environments. Therefore, for use cases demanding higher levels of confidentiality and integrity, modern cryptographic standards such as the Advanced Encryption Standard (AES) or Rivest–Shamir–Adleman (RSA) should be adopted.
- Performance Assessment: The software exhibits excellent performance characteristics, with encryption and decryption operations being completed nearly instantaneously, even for relatively long text inputs. Resource consumption is minimal, and no memory leaks or runtime errors were observed during testing. The program's performance efficiency, combined with its low complexity, renders it suitable for execution on basic computing systems with limited processing power.
- Comparative Analysis: A comparative analysis of the two substitution methods reveals the trade-offs between simplicity and security. The Caesar cipher offers ease of implementation and minimal overhead but suffers from severe cryptographic weaknesses. On the other hand, the custom random substitution table introduces a more complex structure that considerably elevates security, albeit still falling short of contemporary standards for secure communication. This trade-off highlights the importance of aligning encryption method selection with the specific security requirements of the application context.
- In conclusion, the developed program successfully achieves the core objectives of accuracy, usability, and performance. It serves as a functional demonstration of classical encryption techniques and provides a valuable educational platform for understanding substitution ciphers. However, its limitations in security underscore the need for integration with modern cryptographic algorithms when deployed in real-world applications demanding stronger protections. Future enhancements may include the implementation of hybrid encryption schemes or the adoption of standardized cryptographic libraries to elevate security while maintaining usability.

8. Conclusion

This study has detailed the development and implementation of an educational simulation program for secure text encryption and decryption using the Substitution Cipher algorithm. The simulation was designed to incorporate two distinct encryption techniques: the classical Caesar Shift cipher and a user-defined random substitution table comprising 26 unique characters. By integrating both approaches, the program offers a comprehensive platform for learners to explore the operational principles, strengths, weaknesses, and real-world applicability of substitution-based encryption techniques.

Experimental evaluations confirm that the program functions with a high degree of accuracy, preserving the integrity of the original message throughout the encryption-decryption cycle. The decrypted output consistently matches the original plaintext, demonstrating the correctness and reliability of the implemented algorithm. Furthermore, the graphical user interface (GUI) enhances the usability of the program, enabling users to perform encryption and decryption tasks with minimal effort and providing immediate feedback, thereby supporting a hands-on learning experience.

In terms of cryptographic strength, the Caesar Shift method, due to its predictable and fixed-character shift, exhibits limited resistance to frequency analysis and brute-force attacks, rendering it unsuitable for secure applications. In contrast, the use of a randomized substitution table significantly enhances the complexity of decryption attempts, offering improved security. However, it must be noted that even this enhanced substitution method remains vulnerable to advanced cryptanalysis techniques and does not meet the standards required for modern high-security systems.

Through this practical exercise, learners not only develop a solid understanding of the fundamental workings of substitution ciphers but also gain critical insights into the algorithm's inherent limitations and scope of application. This fosters an appreciation of the trade-off between simplicity and security in cryptographic design.

Looking ahead, the educational value of this practice can be further enhanced by integrating more robust encryption standards such as the Advanced Encryption Standard (AES) or Rivest–Shamir–Adleman (RSA) algorithm. Additionally, the simulation may benefit from the inclusion of hybrid encryption techniques or authentication mechanisms, thereby aligning it more closely with contemporary security practices and expanding its pedagogical utility in information security education.

9. References

1. Setiadi DRIM, Salam A, Rachmawanto EH, Sari CA. Improved color image encryption using modified modulus substitution cipher, dual random key and chaos method. Comput Sist. 2023;27(2).
2. Veerasignam V, Harun NZ. A security level comparison of Caesar cipher, columnar transposition cipher and row transposition cipher in Tamil messages. Appl Inf Technol Comput Sci. 2023;4(1):92–108.
3. Nguyễn HH, Đặng QGB, Kim DY, Namgoong Y, Noh SC. Three steps polyalphabetic substitution cipher practice model using Vigenere table for encryption. Converg Secur J. 2022;22:33–9.
4. Python cryptography: Implementing a simple substitution cipher [Internet]. CodeFiner; 2024 [cited 2025 Apr 14]. Available from: https://codefiner.com
5. SC-SA: Byte-oriented lightweight stream ciphers based on S-box substitution. Symmetry. 2023;16(8):1051.
6. Hilton R. Automated cryptanalysis of monoalphabetic substitution ciphers using stochastic optimization algorithms [dissertation]. Colorado (USA): University of Colorado; 2012.

7. Jain A, Chaudhari NS. A new heuristic based on the cuckoo search for cryptanalysis of substitution ciphers. In: Proceedings of the 22nd International Conference on Neural Information Processing. Springer; 2015. p. 206–15.
8. Jadon SS, Sharma H, Kumar E, Bansal JC. Application of binary particle swarm optimization in cryptanalysis of DES. In: Proceedings of the International Conference on Soft Computing for Problem Solving. Springer; 2012. p. 1061–71.
9. Matthews RA. The use of genetic algorithms in cryptanalysis. Cryptologia. 1993;17(2):187–201.