WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Developing a Practical Exercise for Secure Encryption and Decryption Using the Substitution Cipher Algorithm

Truong Van Thu

Faculty of National Defense - Security Education, HaNoi University of Science and technology, Viet Nam

A B S T R A C T

Information security is a crucial aspect of communication and data safety, especially in scenarios requiring the protection of sensitive information, such as operational commands. This paper presents the development of a practical exercise simulating the encryption and decryption process using a substitution cipher method, where a secret key is defined within a fixed character substitution table.

The practical exercise focuses on encrypting and decrypting the operational command "XUAT KICH VAO GIO GIAO THUA" by replacing each character in the original text with its corresponding character in the substitution table. The implementation in MATLAB allows students to directly program, observe the encryption–decryption process, and verify the accuracy of the algorithm.

The primary objective of this exercise is to help students understand the principles of substitution encryption, assess the security level of this method, and recognize its strengths and limitations when applied in real-world scenarios. Experimental results indicate that this approach enhances students' programming skills, analytical thinking in algorithm design, and the ability to apply information security techniques to practical problems.

Keywords: Encryption, Decryption, Substitution Cipher, Information Security

Introduction

In the context of increasing concerns over information security, data encryption remains one of the most effective methods for protecting information. Traditional encryption algorithms, particularly the substitution cipher, continue to play a significant role in cryptography education and research [4].

Substitution cipher encryption is a classical method in which each character in the original text is replaced with another character based on a predefined rule. Common variations of this method include Caesar cipher, Vigenère cipher, and multi-alphabet encryption systems [3]. Despite its simple structure, this algorithm is still applied in various fields such as personal data protection, military communication, and identity recognition systems [1, 5]. Notably, in military environments, substitution cipher encryption can help safeguard critical operational commands against decryption by adversaries.

Several studies have proposed enhancements to improve the security of substitution cipher encryption. For example, integrating substitution encryption with chaos-based methods increases cryptanalysis difficulty [1], while applying a three-step Vigenère table provides a clearer illustration of the algorithm's operation [3]. Additionally, some research has evaluated the security levels of different substitution encryption techniques, such as Caesar cipher, columnar transposition, and row transposition, to determine the most optimal algorithm for specific use cases [2].

Besides encryption algorithm research, cryptanalysis methods also play a crucial role in assessing the security of encryption systems. Modern technologies such as swarm intelligence optimization, cuckoo search, and genetic algorithms have been successfully applied in decrypting substitution cipher systems [6, 7]. These studies not only help test the security of encryption techniques but also contribute to the development of more robust encryption methods.

Based on these practical needs, this paper proposes the development of a simulation program for encryption and decryption using a substitution cipher, employing a predefined secret key in a substitution table. The program is designed to protect a critical operational command, specifically "XUAT KICH VAO GIO GIAO THUA", by encrypting and decrypting it using a predefined substitution rule.

The development of this program not only serves as a practical verification of the effectiveness of substitution cipher encryption but also provides a valuable tool for teaching and researching cryptography. The following sections of this paper will provide a detailed discussion of the encryption algorithm, the implementation of the simulation program, and an evaluation of the security level of this method.

2. Problem Statement

The objective is to develop a simulation program for encryption and decryption using the substitution cipher method with a predefined secret key in a substitution table to protect the operational command: "XUAT KICH VAO GIO GIAO THUA"

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. Steps for Implementation

To develop a simulation program for the character substitution encryption and decryption method using a secret key in a substitution table, students should follow the steps outlined below:

Step 1: Define the Substitution Table

Description of the substitution table: Provide the given substitution table, where each character in the original table (A-Z) is replaced by a corresponding character from the substitution table defined by the secret key.

Secret key: This key will serve as the substitution table, determining how characters in the original text are mapped to the encrypted text and vice versa.

Step 2: Preprocess the Text

Remove whitespace: In this step, whitespace and unnecessary characters should be removed to prepare the data for encryption.

Store information about whitespace: To restore the original text after decryption, we will store the positions of the whitespace in the original text. These positions will be used during decryption to insert whitespace back into the correct locations in the text.

Step 3: Encrypt the Text

Apply the character substitution algorithm: Each character in the processed text will be substituted with the corresponding character from the substitution table.

Step 4: Decrypt the Text

Decrypt using the reverse substitution table: For decryption, we use the reverse substitution table, replacing characters in the encrypted text with their corresponding characters in the original character table.

Restore whitespace: After decryption, the list of whitespace positions stored during the preprocessing step will be used to insert whitespace back into the correct locations in the text. This ensures the original text is reconstructed.

Step 5: Develop a User Interface (GUI)

Simple user interface: Students need to design a user interface to easily input the text to be encrypted or decrypted, perform encryption/decryption, and display the results.

Key functions of the interface include:

Input text: Allow the user to enter the text to be encrypted or decrypted.

Encryption/Decryption: Perform encryption or decryption when the appropriate function is selected.

Display results: After encryption or decryption, display the results clearly on the interface.

Step 6: Test and Evaluate the Results

Compare results: Verify the accuracy of the program by comparing the original text with the decrypted text. If the decrypted result matches the original text, the algorithm is functioning correctly.

Evaluate the security of the method: Analyze the security of character substitution encryption in protecting information. This step is crucial for evaluating the method's resistance to attacks and its overall security.

Through these steps, students will not only build a complete program simulating the character substitution encryption and decryption process but also gain a deeper understanding of critical aspects of information security, including data encryption, restoration of original data, and evaluation of the security of applied methods in practice.

6. Program Code

```
function SUBSTITUTION_CIPHER
    % Create the main interface
    fig = figure ('Name', 'SUBSTITUTION CIPHER', 'Position', [500 200 650 450]);
    % Standard size
    labelWidth = 200;
    boxWidth = 420;
    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
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 → F, ⋯).

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

Fig.2. Results of Encoding and Decoding Using the Custom 26-Letter Character Substitution Table (e.g., QWERTYUIOPASDFGHJKLZXCVBNM)

7.2 Evaluation of Effectiveness

This practice uses the Substitution Cipher algorithm to encrypt and decrypt text. In terms of security, when applying the default Caesar Shift character substitution table, the level of security is not high because the algorithm follows a fixed shift rule, making it vulnerable to cryptanalysis through frequency analysis of characters. However, when using a custom substitution table with 26 random letters, the security is significantly improved as the process of brute-forcing the substitution table becomes more difficult.

Regarding accuracy, the program ensures that the encryption and decryption process is performed with absolute precision. The decrypted text fully matches the original text, with no data loss or character distortion. Experimental testing shows that the program runs stably without errors.

The program also demonstrates excellent automation, allowing users to input text and quickly perform encryption and decryption with the press of a button. Additionally, the feature to clear all data allows users to easily conduct multiple trials without restarting the interface.

In terms of practical application, the substitution cipher algorithm can be applied in simple systems that require internal information protection. However, for applications requiring a higher level of security, this algorithm is not strong enough and should be replaced with more advanced methods such as AES or RSA.

Regarding performance, the program operates quickly even when processing long texts. The interface is intuitive and user-friendly, making it easy for users to operate without needing a deep understanding of the algorithm. When comparing the two encryption methods, using the Caesar Shift table is simple but vulnerable to cracking, while the custom substitution table significantly improves security, though it is still not strong enough to guarantee absolute safety.

In summary, the program meets experimental requirements, ensuring accuracy, ease of use, and applicability in situations requiring basic encryption. However, to enhance security, modern encryption methods should be integrated for practical use.

8. Conclusion

This paper has presented the process of developing a secure encryption and decryption practice based on the character substitution algorithm. Through the implementation of the program with two methods: the default Caesar Shift character substitution table and the custom 26-character substitution table, the practice helps learners understand the operating principles, advantages, disadvantages, and practical applications of this algorithm.

Experimental results show that the program operates accurately, ensuring data integrity after encryption and decryption. Additionally, the intuitive interface and easy-to-use controls enable users to perform experiments quickly. However, in terms of security, the Caesar Shift method has a low level of security, while the method using the custom substitution table increases the difficulty of decryption but is still not strong enough to meet the requirements for high-level security.

From this practice, learners not only grasp the working mechanism of the substitution cipher algorithm but also gain an overview of its limitations and practical applications. In the future, the practice could be expanded by integrating stronger encryption algorithms such as AES or RSA or by combining them with other security methods to increase its practicality and enhance teaching effectiveness.

References

Setiadi, D. R. I. M., Salam, A., Rachmawanto, E. H., & Sari, C. A. (2023). Improved Color Image Encryption using Modified Modulus Substitution Cipher, Dual Random Key and Chaos Method. *Computación y Sistemas*, 27(2).

Veerasignam, V., & Harun, N. Z. (2023). A Security Level Comparison of Caesar Cipher, Columnar Transposition Cipher and Row Transposition Cipher in Tamil Messages. *Applied Information Technology And Computer Science*, 4(1), 92-108.

Nguyễn, H. H., Đặng, Q. G. B., Kim, D. Y., Namgoong, Y., & Noh, S. C. (2022). Three Steps Polyalphabetic Substitution Cipher Practice Model using Vigenere Table for Encryption. *Convergence Security Journal*, 22, 33-39.

Python Cryptography: Implementing a Simple Substitution Cipher. (2024). *CodeFiner*.

SC-SA: Byte-Oriented Lightweight Stream Ciphers Based on S-Box Substitution. (2023). *Symmetry*, 16(8), 1051.

Hilton, R. (2012). Automated Cryptanalysis of Monoalphabetic Substitution Ciphers Using Stochastic Optimization Algorithms. *Doctoral dissertation, Department of Computer Science and Engineering, University of Colorado, USA*.

Jain, A., & Chaudhari, N. S. (2015). A New Heuristic Based on the Cuckoo Search for Cryptanalysis of Substitution Ciphers. *In Proceedings of the 22nd International Conference on Neural Information Processing* (pp. 206–215). Springer.

Jadon, S. S., Sharma, H., Kumar, E., & Bansal, J. C. (2012). Application of Binary Particle Swarm Optimization in Cryptanalysis of DES. *In Proceedings of the International Conference on Soft Computing for Problem Solving* (pp. 1061–1071). Springer.

Matthews, R. A. (1993). The Use of Genetic Algorithms in Cryptanalysis. *Cryptologia*, 17(2), 187–201.