Project Report

On

Building a File Encryption & Decryption Tool

Submitted To

Amity University

In partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

(***Computer Science And Business Systems***)

By

Arul Gupta

(A023165822020)

May 2025 - June 2025

Abstract

For a long time, cryptography has been considered a key pillar in secure communication. From the hieroglyphic encryptions of ancient Egypt to the quantum algorithms of 21st-century applications, the science of securing information has undergone a massive transformation [1]. The transposition-like techniques still have much to teach us, especially in an educational setup. The methods teach in a concrete manner the cryptographic principles of confusion and diffusion [3]. One method that is ingenious enough and clear enough to be used at least as a demonstration model is the columnar transposition cipher [4].

This report undertakes a systematic and exhaustive study of transposition cryptography with a focus on the columnar transposition cipher. It covers the historical significance, theoretical basis, algorithmic principle, and implementation of the method. Then it analyzes substitution and transposition cryptography [5], comparing monoalphabetic and polyalphabetic substitution methods [6], and then extending into the realm of more complicated transposition algorithms. The report also covers the design and implementation of a simple-to-use encryption and decryption tool written in Python with the Tkinter GUI library, motivated by modern teaching tools [7]. This section discusses design choices, software architecture, algorithm workflows, and interface functionalities, presenting sample outputs and a reflection on the overall project results. This balanced, multifaceted approach bridges theoretical knowledge with practical application in an equitable fashion.

Introduction

Global digital communication expands every day, making secure data transmission critically fundamental. Concern for guarding information from unauthorized parties has been a requirement since ancient times. Cryptography, a science that focuses on protecting information, began in ancient civilizations. One of the earliest recorded instances of cryptography was around 1900 BCE in Egypt, when non-standard hieroglyphs were used for code [8]. The Spartans used a device called the scytale, which enclosed a strip of parchment that would wrap around a cylinder- the scytale would encrypt messages. The Romans are well known for using the Caesar cipher, which is a basic substitution of shifting letters of the alphabet used for private correspondence [9].

Cryptographic practices evolved significantly over the years. More sophisticated techniques were created during the Renaissance as well as the World Wars. In the 20th century, mechanical encryption devices like the Enigma machine, which was used commonly in World War II [10], [11], began to appear. Such inventions set the foundation for today's digital cryptography systems [12].

In this historical framework, transposition cryptography is one of the techniques of primary importance. In contrast to substitution methods, which change certain characters into others, transposition methods keep the characters the same, but rearrange them, thus making it impossible to interpret the message without the proper key [13].

This report studies the transposition cipher, focusing on columnar transposition. We will start with the basics of cryptography, cover substitution ciphers and their modifications, and then present the workings of transposition methods. The tool designed in this project allows users to encrypt and decrypt data via a graphical interface developed in Python using the Tkinter library [14].

Theory

Basics of Cryptography

Cryptography is both a scientific discipline and an art that involves securing communication through encoding (known as encryption) and decoding (known as decryption). Its main goal is to safeguard secrets about information, and its integrity and authenticity from unauthorized viewing [2]. Cryptography as a discipline has ancient roots, as there is evidence of secret writing used in the Egyptian, Greek, and Roman cultures [1]. Modern classical cryptography stems from these early practices.

Classical Cryptography encompasses encryption methods that were actively used long before computers were invented. Most of these techniques are manual or mechanical, with some of the most famous ciphers in history. Notable examples are the Caesar cipher, the Atbash cipher, the Scytale transposition cipher by the Spartans, and the Vigenère cipher [9], [10], [11]. Two primary techniques tend to underlie most classical cryptography.

Substitution: This process involves the replacement of a character in a message with another character based on an agreed-upon rule or key. For instance, in the Caesar cipher, each letter in the alphabet is replaced by a letter that appears a certain number of places ahead of it in the alphabet [8], [9].

Transposition: All characters are preserved in this method. Instead of replacing characters as in substitution, this approach consists of shuffling the characters of the plaintext in order to create the ciphertext. While the characters are changed, their identity remains the same, only their position is changed to conceal the message [13].

Both substitution and transposition have in common that a cipher (method or algorithm) is used to perform a transformation with a key (the item used to control encryption and decryption) able to perform both encryption and decryption in classical cryptography. Although classical cryptographic algorithms may appear to be straightforward in comparison to today's modern algorithms, they introduce fundamental concepts such as the confusion and diffusion of keys, principles which were formalized by Claude Shannon [3].

Encryption and Decryption

Encryption involves the use of an algorithm and a key to encode meaningful data referred to as plaintext and transform it into unreadable ciphertext [2]. Obtaining the plasmid's nucleotide sequence, or retrieving encrypted data, by employing the appropriate method, also known as key, is the reverse of encryption and is known as decryption.

Symmetric cryptography uses a single key for both encryption and decryption processes, while asymmetric cryptography employs a different key for them [2]. The security of a cryptographic system relies on the algorithm used as well as the key secrecy. Having an inadequate algorithm or poorly managed keys compromises the system's overall security [5].

Substitution cryptography replaces each element in the plaintext with another character or symbol. Monoalphabetic substitution, like the Caesar cipher uses a single fixed mapping. It is comparatively fast but is vulnerable to frequency analysis [9]. Vigenere cipher, a polyalphabetic variant, offers better security by rotating through multiple cipher alphabets to conceal repetitive patterns in ciphertext until statistical attacks, such as Kasiski examination, come into play [10]. Transposition ciphers do not alter the characters of the plaintext but rather rearrange them. In Shannon’s terminology, these ciphers achieve diffusion by spreading character dependencies throughout the ciphertext. Amongst others are the rail fence, route, and columnar transposition ciphers [13].

Columnar Transposition Cryptography

A message in this cipher is encoded by placing its characters in a specified order, taking cues from a keyword and a set columnar system. Transposition ciphers use the same characters from the plaintext, but swap their positions [15]. The plaintext is arranged in rows and matched with columns based on the keyword specified. After that, the columns are listed in alphabetical order according to the keyword, and the ciphertext is read starting from the top left of each column.

Being simple to follow and mildly secure in encryption, the columnar transposition cipher is a popular one and has been studied a lot. It continues to be relevant for showing how diffusion and the key dependency work in cryptography [16]. CrypTool and other learning platforms like it are often used in schools and for e-learning to help students understand both classical and modern cryptography [17]. Having these features makes it suitable for use in learning and research.

As described by ChatGPT [18], the columnar transposition cipher involves several structured steps:

Step 1: Preprocessing the Message
The input plaintext message is first stripped of spaces and special characters to ensure uniformity. This clean string is then converted to uppercase to avoid complications arising from case sensitivity. For instance, the message "Meet me at the park" becomes "MEETMEATTHEPARK".

Step 2: Determining the Grid Structure
A keyword or cipher key is selected, such as "CIPHER". Each character in the key is assigned a numeric position based on alphabetical sorting. The alphabetical order of "CIPHER" is C (1), E (2), H (3), I (4), P (5), R (6). This results in a key order of [1, 4, 5, 3, 2, 6]. The length of the key determines the number of columns in the grid.

Step 3: Padding the Message
The number of characters in the message may not always fill the grid completely. Suppose the cleaned message has 17 characters and the key length is 6, the number of rows needed is calculated as:

rows = ceil(len(message) / len(key)) = ceil(17 / 6) = 3

The total grid space will be 6 * 3 = 18. Therefore, the message is padded with a filler character (commonly 'X') to reach 18 characters: "MEETMEATTHEPARKXX".

Step 4: Populating the Grid
The message is written row-by-row into the grid:

C (1)	**I (4)**	**P(5)**	**H(3)**	**E(2)**	**R(6)**
M A A	E T R	E T K	T H X	M E X	E P X

(Note: The first row displays the key for reference.)

Step 5: Generating Ciphertext
The columns are then rearranged and read in the order determined by sorting the keyword

alphabetically. For "CIPHER", the sorted order becomes C-E-H-I-P-R, or column indices [1, 2, 3, 4, 5, 6]. Reading down these columns produces the ciphertext by concatenating the characters column-wise:

C(1)	**E(2)**	**H(3)**	**I (4)**	**P(5)**	**R(6)**
M A A	M E X	T H X	E T R	E T K	E P X

- Column 1: M, A, A
- Column 2: M, E, X
- Column 3: T, H, X
- Column 4: E, T, R
- Column 5: E, T, K
- Column 6: E, P, X

Final ciphertext: "**MAAMEXTHXETRETKEPX**"

Step 6: Decryption Process
To decrypt, the receiver must know the key. The length of the key and message helps in reconstructing the number of rows and determining how many characters fit in each column. The ciphertext is divided based on the column order into segments and filled column-wise into an empty grid. Once the grid is reconstructed, the original message is read row-by-row, and the padding character 'X' is removed from the end if present.

C(1)	M A A
E(2)	M E X
H(3)	T H X
I(4)	E T R
P(5)	E T K
R(6)	E P X

The rows are then rearranged and read in the order determined by sorting the keyword alphabetically.

C(1)	M A A
I(4)	E T R
P(5)	E T K
H(3)	T H X
E(2)	M E X
R(6)	E P X

- Column 1: M, E, E, T, M, E
- Column 2: A, T, T, H, E, P
- Column 3: A, R, K, X, X, X

Remove the trailing X's.

Final ciphertext: "**MEETMEATTHEPARK**"

This technique increases the complexity of deciphering the message without the key, as the character sequence is scrambled in a systematic but non-obvious manner. It also preserves the original character frequency, making statistical attacks less effective in some scenarios.

Despite its strengths, the columnar transposition cipher has limitations. It is vulnerable to brute force if the attacker can guess the key length, especially since the number of possible permutations for a key of length n is n!. For small key lengths, this is computationally feasible. Thus, while effective for low-security applications and historical understanding, it is not suitable for securing sensitive modern communications.

Advantages

One of the most notable advantages of the columnar transposition cipher is how easy it is to implement. It sidesteps advanced operations, which makes it appealing to novices and a great introductory tool for teaching concepts like permutation and diffusion [19]. Encryption maintaining characters helps learners perceive encryption as reordering rather than substitution [17]. Moreover, since characters are not modified, frequency analysis is more difficult than in monoalphabetic substitution, although possible [20].

Another pro is its flexibility. Its ciphertext output can be greatly changed by modifying key length or adjusting grid size and shape, thus increasing complexity. This adaptability, paired with its grid representation, makes it useful in the classroom [17].

Disadvantages

While effective for classroom use, the columnar transposition cipher is weak by today's standards in cryptography. It is particularly susceptible to brute force attacks, such as when the attacker can make an educated guess about the key length [16], [15]. Transposition ciphers are also vulnerable to cryptanalysis with multiple copies of the ciphertext or when the key cycles through values multiple times [21].

In addition, hybrid methods that integrate substitution with modular arithmetic or publicly accessible systems increase the difficulty of deciphering transposed text, which columnar transposition does not possess [21]. Therefore, with regards to its usefulness in teaching and low-risk scenarios, it is unfit for safeguarding sensitive data [23].

Code

Modules Used

The modules **tkinter**, **ttk**, **filedialog**, **messagebox**, and **scrolledtext** are a part of the standard GUI library in Python. Tkinter is the main module that constructs the graphical interface of the program and it is implemented with widgets using the ttk module which provides enhanced visual styling. The function of filedialog is that it provides selection of dialogs for files, messagebox allows alerts and confirmation popups with various options, and integrating scrolledtext with other text modules provides areas displaying extended content, supporting scrolling and enabling enhanced text navigation [24].

The **math** module includes multiple functions based on a range of mathematical operations. This project utilizes its ceil() function for calculating the amount of rows in the transposition cipher's grid [25].

The **os** module makes it possible to interact with the operating system for checking the validity of paths and managing directories for the storage of the outputs whether encrypted or decrypted [26]. The additional functionality comes from the **datetime** module, which serves to provide timestamps to files to be generated, ensuring the generated copies do not overwrite existing copies while making organized references based on an indexed chronology [27].

Classes and Functions

1. TranspositionCipher

This class manages the user interfaces within Tkinter as it creates and handles the GUI. It allows the user to utilize the encryption/decryption tool using buttons, entry boxes, radio buttons as well as text areas. Moreover, it facilitates the exchange of data between the interface and the cryptographic logic.

- generate_key_order(cipher_key): This function is very important in deciding the order in which the columns of the message grid will be read. It first intersects the letters of the key with the alphabet. While performing the intersection, it removes duplicate letters from the key but maintains the order. Finally, it sorts the set of letters and assigns a numeric value to each letter. The column reading order is useful for both encryption and decryption processes.
- encrypt(message, cipher_key): This method works with the plain text and a specific cipher key to encrypt and produce the corresponding ciphertext. The first step is always cleaning the message by getting rid of unnecessary characters before capitalizing it. After calculating the grid size, if necessary, the message is padded with 'X' characters. The characters are first arranged row by row in a grid based on the length of the key. Subsequently, they are read column by column based on the generated key order to yield the ciphertext.
- decrypt(cipher_text, cipher_key): This method undoes the subsequent encryption of the message. It fills in the grid as per the key order by the characters in the ordered ciphertext during the specified period. Once filled, it retrieves the original message by traversing it horizontally, omitting unnecessary characters engraved at the end.

2. TranspositionGUI

This class is responsible for the development and control of the graphical user interface with Tkinter. It allows the user to control the encryption/decryption tools via buttons, entry fields, radio buttons, and text areas. It manages the stages of data processing, communication, and cryptographic logic.

- __init__(root): The operation sets up the class GUI by initializing the root window alongside its width and height, default parameters of the window, such as operation mode: encrypt or decrypt, and constructors for GUI components.
- setup_gui(): This function arranges and creates all input panels, output panel, toggle, and file browsing buttons, status messages to obey the functioning guidelines of a thoroughly assisting layout designed for aiding the user in every guided step.
- process_text(): This approach serves as a mediator between the interface and the machine logic. It fetches the data from the user (be it in the form of text or a file), checks the key, calls the encryption and decryption method accordingly, and shows the output in the output panel. It also takes care of error notifications as well as updating status feedback.
- save_output(): This is the method which allows a user to save the output text in a file, whether it be encrypted or decrypted. With the help of the filedialog box, the user can select a path and filename, and the system will append the current date and time to the file name to avoid unintentional overwriting. Additionally, this method uses exception handling for file input/output.

Algorithm for Encryption and Decryption

Before diving into the source code, let us take a look at the algorithm to understand the logic that drives the columnar transposition cipher used in this project.

Algorithm 1 Encrypt(message, cipher_key)

```
1:  $(unique\_key, key\_order) \leftarrow$ generate_key_order($cipher\_key$)
2:  $cols \leftarrow$ length of $unique\_key$
3:  $message \leftarrow$ remove spaces from message and convert to uppercase
4:  $rows \leftarrow \lceil$length of $message/cols\rceil$
5:  $padded\_message \leftarrow$ pad message with 'X' to make its length $rows \times cols$
6:  $grid \leftarrow$ empty list
7:  for $i = 0$ to $rows - 1$ do
8:      $row \leftarrow [\,]$
9:      for $j = 0$ to $cols - 1$ do
10:         $index \leftarrow i \times cols + j$
11:         $row.append(padded\_message[index])$
12:     end for
13:     $grid.append(row)$
14: end for
15: $cipher\_text \leftarrow$ ""
16: for each $col\_index$ in $key\_order$ do
17:     for $row = 0$ to $rows - 1$ do
18:         $cipher\_text \leftarrow cipher\_text + grid[row][col\_index]$
19:     end for
20: end for
21: return $cipher\_text$
```

Fig1. Encryption Algorithm

When the key is “CIPHER”, the text is encrypted by first applying a series of transformations to the input. In this case, the first step is to erase all characters that are not letters, and transform the letters to uppercase. The next step involves deriving the column order for encryption from the key. This process entails first preserving the order of the characters from the key and then removing duplicates. Subsequently, all of the letters are ordered in an increasing manner and assigned a number starting with one. These numbers will be the order in which the columns of the grid will be read and these columns will be filled with the corresponding message. Columns are created until column number equal to the length the input keyword is reached. The number of rows is calculated with the formula:: ceil(length of the input message divided by the length of the keyword). If padding is needed, 'X' is is used as a fill character. Finally, the rows of the grid will be read sequentially, starting from the top row. The order has to follow the key indices which have been ordered from lowest to highest. The string that results from these operations is referred to as ciphertext.

Algorithm 2 Decrypt(cipher_text, cipher_key)

```
1: $(unique\_key, key\_order) \leftarrow$ generate_key_order($cipher\_key$)
2: $cols \leftarrow$ length of $unique\_key$
3: $rows \leftarrow \lceil$length of $cipher\_text/cols\rceil$
4: $total\_chars \leftarrow$ length of $cipher\_text$
5: $chars\_per\_col \leftarrow total\_chars \div cols$
6: $extra\_chars \leftarrow total\_chars \bmod cols$
7: $grid \leftarrow$ 2D list of empty strings with dimensions $rows \times cols$
8: $char\_index \leftarrow 0$
9: for each $col\_index$ in $key\_order$ do
10:     if $col\_index < extra\_chars$ then
11:         $col\_length \leftarrow chars\_per\_col + 1$
12:     else
13:         $col\_length \leftarrow chars\_per\_col$
14:     end if
15:     for $row = 0$ to $col\_length - 1$ do
16:         $grid[row][col\_index] \leftarrow cipher\_text[char\_index]$
17:         $char\_index \leftarrow char\_index + 1$
18:     end for
19: end for
20: $decrypted \leftarrow$ ""
21: for $row = 0$ to $rows - 1$ do
22:     for $col = 0$ to $cols - 1$ do
23:         if $grid[row][col] \neq$ "" then
24:             $decrypted \leftarrow decrypted + grid[row][col]$
25:         end if
26:     end for
27: end for
28: return $decrypted$ with trailing 'X' characters removed
```

Fig2. Decryption Algorithm

The decryption begins by performing preprocessing steps, this includes splitting columns by the length of the key and rows as row = ceil(length of ciphertext)/length of key. Then, adding duplicates of letters to the key is marked with an 'X' while characters are added to the key in alphabetical order when duplicates aren't added. Using the length of the ciphertext, the number of columns is calculated to be first *len(ciphertext) mod len(key)*. Reading the text in a vertical column manner results in filling the grid, bottom up and top down gives a complete grid. Reading the grid in a row-wise manner provides plain text and finally padding characters ‘X’ added during encryption are stripped to get the core message which is the final outcome of the decryption.

Source Code

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox, scrolledtext
import math
import os

class TranspositionCipher:
    @staticmethod
    def generate_key_order(cipher_key):
        """Generate column order based on alphabetical sorting of cipher key"""
        # Remove duplicates while preserving order, convert to uppercase
        seen = set()
        unique_key = ''.join(char.upper() for char in cipher_key if char.upper() not in seen and not
seen.add(char.upper()) and char.isalpha())

        if len(unique_key) < 2:
            raise ValueError("Cipher key must contain at least 2 unique letters")

        # Create list of (character, original_index) pairs
        char_index_pairs = [(char, i) for i, char in enumerate(unique_key)]

        # Sort by character to get alphabetical order
        sorted_pairs = sorted(char_index_pairs, key=lambda x: x[0])

        # Extract the order (which original positions come in alphabetical order)
        key_order = [pair[1] for pair in sorted_pairs]

        return unique_key, key_order

    @staticmethod
    def encrypt(message, cipher_key):
        """Encrypt message using transposition cipher with given cipher key"""
        # Generate key order from cipher key
        unique_key, key_order = TranspositionCipher.generate_key_order(cipher_key)
        cols = len(unique_key)

        # Remove spaces and convert to uppercase for consistency
        message = message.replace(' ', '').upper()

        # Calculate number of rows
        rows = math.ceil(len(message) / cols)
        # Pad message with 'X' if necessary
        padded_message = message.ljust(rows * cols, 'X')

        # Create grid and fill it row by row
        grid = []
        for i in range(rows):
            row = []
            for j in range(cols):
                idx = i * cols + j
                row.append(padded_message[idx])
            grid.append(row)

        # Read columns in the order specified by key_order
        cipher_text = ''
        for col_index in key_order:
            for row in range(rows):
                cipher_text += grid[row][col_index]

        return cipher_text

    @staticmethod
    def decrypt(cipher_text, cipher_key):
        """Decrypt cipher text using transposition cipher with given cipher key"""
        # Generate key order from cipher key
        unique_key, key_order = TranspositionCipher.generate_key_order(cipher_key)
        cols = len(unique_key)
        rows = math.ceil(len(cipher_text) / cols)

        # Calculate how many characters should be in each column
        total_chars = len(cipher_text)
        chars_per_col = total_chars // cols
        extra_chars = total_chars % cols

        # Create empty grid
        grid = [['' for _ in range(cols)] for _ in range(rows)]

        # Fill grid column by column in key order
        char_index = 0
        for col_index in key_order:
            # Determine how many characters this column should have
            col_length = chars_per_col + (1 if col_index < extra_chars else 0)
```

```python
            for row in range(col_length):
                if char_index < len(cipher_text):
                    grid[row][col_index] = cipher_text[char_index]
                    char_index += 1

        # Read grid row by row to get original message
        decrypted = ''
        for row in range(rows):
            for col in range(cols):
                if grid[row][col] != '':
                    decrypted += grid[row][col]

        # Remove padding 'X' from the end
        return decrypted.rstrip('X')

class TranspositionGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Transposition Cipher Tool")

        # Make window fixed size and non-resizable
        self.root.geometry("800x600")
        self.root.resizable(False, False)

        # Center the window on screen
        self.center_window()

        # Current operation mode
        self.current_mode = tk.StringVar(value="encrypt")

        self.setup_gui()

    def center_window(self):
        """Center the window on the screen"""
        self.root.update_idletasks()
        width = 800
        height = 600
        x = (self.root.winfo_screenwidth() // 2) - (width // 2)
        y = (self.root.winfo_screenheight() // 2) - (height // 2)
        self.root.geometry(f'{width}x{height}+{x}+{y}')

    def setup_gui(self):
```

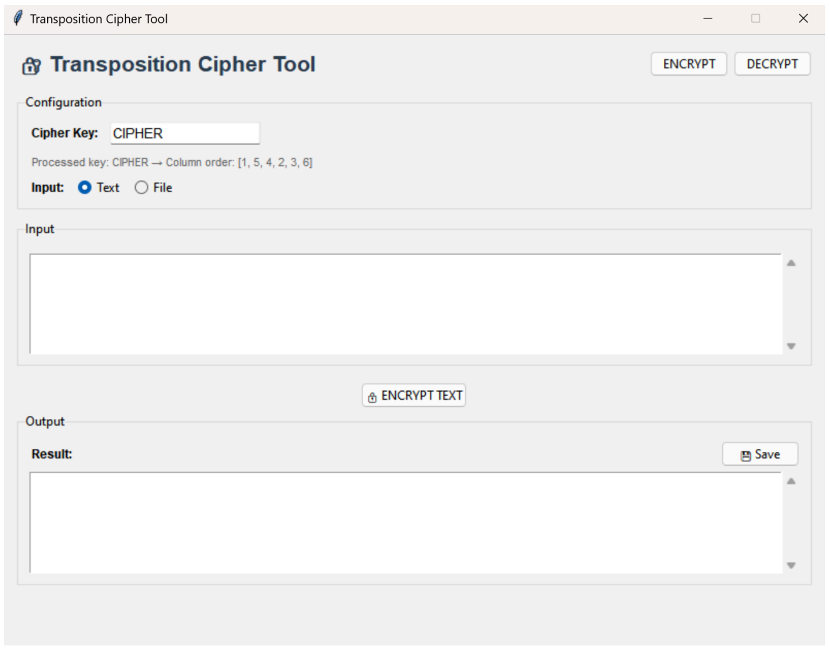
```
# Remove scrollable canvas - use direct frame approach for fixed layout
main_frame = ttk.Frame(self.root, padding="15")
main_frame.pack(fill='both', expand=True)

# Title section - more compact
title_frame = ttk.Frame(main_frame)
title_frame.pack(fill='x', pady=(0, 15))

title_label = ttk.Label(title_frame, text="Transposition Cipher Tool",
                   font=('Arial', 16, 'bold'), foreground='#2c3e50')
title_label.pack(side='left')

# Mode toggle buttons
mode_frame = ttk.Frame(title_frame)
mode_frame.pack(side='right')

self.encrypt_btn = ttk.Button(mode_frame, text="ENCRYPT",
                         command=lambda: self.set_mode("encrypt"))
self.encrypt_btn.pack(side='left', padx=(0, 5))

self.decrypt_btn = ttk.Button(mode_frame, text="DECRYPT",
                         command=lambda: self.set_mode("decrypt"))
self.decrypt_btn.pack(side='left')

# Configuration section - more compact
config_frame = ttk.LabelFrame(main_frame, text="Configuration", padding="10")
config_frame.pack(fill='x', pady=(0, 10))

# Key input - horizontal layout
key_frame = ttk.Frame(config_frame)
key_frame.pack(fill='x', pady=(0, 8))

ttk.Label(key_frame, text="Cipher Key:", font=('Arial', 9, 'bold')).pack(side='left')
self.key_var = tk.StringVar(value="SECRET")
key_entry = ttk.Entry(key_frame, textvariable=self.key_var, width=20, font=('Arial', 10))
key_entry.pack(side='left', padx=(10, 0))

# Key hint - compact
self.key_hint = ttk.Label(config_frame, text="", font=('Arial', 8), foreground='#666666')
self.key_hint.pack(anchor='w')

# Input method selection - horizontal
```

```
method_frame = ttk.Frame(config_frame)
method_frame.pack(fill='x', pady=(5, 0))

ttk.Label(method_frame, text="Input:", font=('Arial', 9, 'bold')).pack(side='left')
self.input_method = tk.StringVar(value="text")

ttk.Radiobutton(method_frame, text="Text", variable=self.input_method,
          value="text", command=self.toggle_input_method).pack(side='left', padx=(10, 0))
ttk.Radiobutton(method_frame, text="File", variable=self.input_method,
          value="file", command=self.toggle_input_method).pack(side='left', padx=(10, 0))

# Input section - reduced height
input_frame = ttk.LabelFrame(main_frame, text="Input", padding="10")
input_frame.pack(fill='both', expand=True, pady=(0, 10))

# File selection frame
self.file_frame = ttk.Frame(input_frame)

file_row = ttk.Frame(self.file_frame)
file_row.pack(fill='x')

ttk.Label(file_row, text="File:", font=('Arial', 9, 'bold')).pack(side='left')

self.file_path_var = tk.StringVar()
file_entry = ttk.Entry(file_row, textvariable=self.file_path_var,
               state='readonly', font=('Arial', 9))
file_entry.pack(side='left', fill='x', expand=True, padx=(5, 5))

ttk.Button(file_row, text="Browse", command=self.browse_file).pack(side='right')

# Text input frame
self.text_frame = ttk.Frame(input_frame)

self.input_text = scrolledtext.ScrolledText(self.text_frame, height=6, wrap=tk.WORD,
                          font=('Consolas', 10))
self.input_text.pack(fill='both', expand=True)

# Action button
action_frame = ttk.Frame(main_frame)
action_frame.pack(pady=5)

self.action_btn = ttk.Button(action_frame, text="ENCRYPT TEXT",
                                     command=self.process_text)
        self.action_btn.pack()

        # Output section - reduced height
        output_frame = ttk.LabelFrame(main_frame, text="Output", padding="10")
        output_frame.pack(fill='both', expand=True, pady=(0, 10))

        # Output header with save button
        output_header = ttk.Frame(output_frame)
        output_header.pack(fill='x', pady=(0, 5))

        ttk.Label(output_header, text="Result:", font=('Arial', 9, 'bold')).pack(side='left')
        ttk.Button(output_header, text="Save", command=self.save_output).pack(side='right')

        self.output_text = scrolledtext.ScrolledText(output_frame, height=6, wrap=tk.WORD,
                                                     font=('Consolas', 10))
        self.output_text.pack(fill='both', expand=True)

        # Status frame - compact
        self.status_frame = ttk.Frame(main_frame)
        self.status_frame.pack(fill='x', pady=(5, 0))

        # Bind key change event
        self.key_var.trace('w', self.update_key_hint)

        # Initialize
        self.set_mode("encrypt")
        self.toggle_input_method()
        self.update_key_hint()

    def update_key_hint(self, *args):
        """Update the key order hint when key changes"""
        try:
            cipher_key = self.key_var.get()
            if cipher_key:
                unique_key, key_order = TranspositionCipher.generate_key_order(cipher_key)
                hint_text = f"Processed key: {unique_key} → Column order: {[i+1 for i in key_order]}"
                self.key_hint.configure(text=hint_text)
            else:
                self.key_hint.configure(text="")
        except Exception:
            self.key_hint.configure(text=" Invalid key - use at least 2 unique letters")
```

```python
def set_mode(self, mode):
    """Set the current operation mode"""
    self.current_mode.set(mode)

    # Update button appearance
    if mode == "encrypt":
        self.encrypt_btn.configure(style='Accent.TButton')
        self.decrypt_btn.configure(style='TButton')
        self.action_btn.configure(text="ENCRYPT TEXT")
    else:
        self.encrypt_btn.configure(style='TButton')
        self.decrypt_btn.configure(style='Accent.TButton')
        self.action_btn.configure(text="DECRYPT TEXT")

    # Clear output when switching modes
    self.output_text.delete("1.0", tk.END)

def toggle_input_method(self):
    """Toggle between text and file input methods"""
    # Hide both frames first
    self.file_frame.pack_forget()
    self.text_frame.pack_forget()

    # Show the selected frame
    if self.input_method.get() == "file":
        self.file_frame.pack(fill='x', pady=(5, 0))
    else:
        self.text_frame.pack(fill='both', expand=True, pady=(5, 0))

def browse_file(self):
    """Open file dialog to select input file"""
    file_path = filedialog.askopenfilename(
        title="Select file to process",
        filetypes=[("Text files", "*.txt"), ("All files", "*.*")]
    )
    if file_path:
        self.file_path_var.set(file_path)

def get_input_text(self):
    """Get input text from either text widget or file"""
    if self.input_method.get() == "file":
        file_path = self.file_path_var.get()
        if not file_path:
            messagebox.showerror("Error", "Please select a file")
            return None

        try:
            with open(file_path, 'r', encoding='utf-8') as f:
                return f.read()
        except Exception as e:
            messagebox.showerror("Error", f"Error reading file: {str(e)}")
            return None
    else:
        text = self.input_text.get("1.0", tk.END).strip()
        if not text:
            messagebox.showerror("Error", "Please enter some text")
            return None
        return text

def validate_key(self):
    """Validate the cipher key"""
    try:
        cipher_key = self.key_var.get().strip()
        if not cipher_key:
            messagebox.showerror("Error", "Please enter a cipher key")
            return None

        # Test if key is valid by trying to generate key order
        unique_key, key_order = TranspositionCipher.generate_key_order(cipher_key)
        return cipher_key

    except ValueError as e:
        messagebox.showerror("Error", str(e))
        return None
    except Exception as e:
        messagebox.showerror("Error", f"Invalid cipher key: {str(e)}")
        return None

def process_text(self):
    """Process text based on current mode"""
    text = self.get_input_text()
    key = self.validate_key()
    if text is None or key is None:
        return

    try:
        cipher = TranspositionCipher()

        if self.current_mode.get() == "encrypt":
            result = cipher.encrypt(text, key)
            success_msg = "Text encrypted successfully!"
        else:
            result = cipher.decrypt(text, key)
            success_msg = "Text decrypted successfully!"

        self.output_text.delete("1.0", tk.END)
        self.output_text.insert("1.0", result)

        # Show success message
        self.show_status_message(success_msg, "success")

    except Exception as e:
        messagebox.showerror("Error", f"Operation failed: {str(e)}")

def show_status_message(self, message, msg_type="info"):
    """Show status message"""
    # Clear existing status messages
    for widget in self.status_frame.winfo_children():
        widget.destroy()

    # Choose color based on message type
    if msg_type == "success":
        bg_color = '#d4edda'
        fg_color = '#155724'
        icon = "✅"
    elif msg_type == "error":
        bg_color = '#f8d7da'
        fg_color = '#721c24'
        icon = "❌"
    else:
        bg_color = '#d1ecf1'
        fg_color = '#0c5460'
        icon = "ℹ️"
```

```python
        # Create status message
        status_label = tk.Label(self.status_frame, text=f"{icon} {message}",
                               font=('Arial', 10, 'bold'),
                               bg=bg_color, fg=fg_color,
                               padx=15, pady=8, relief='solid', borderwidth=1)
        status_label.pack(fill='x')

        # Remove after 3 seconds
        self.root.after(3000, lambda: status_label.destroy())

    def save_output(self):
        """Save output text to file - FIXED VERSION"""
        try:
            # Get output text and strip only trailing whitespace to preserve formatting
            output = self.output_text.get("1.0", tk.END)
            # Remove the automatic newline that tkinter adds at the end
            if output.endswith('\n'):
                output = output[:-1]

            if not output or output.strip() == "":
                messagebox.showerror("Error", "No output to save")
                return

            # Suggest filename based on mode and current timestamp
            import datetime
            mode = self.current_mode.get()
            timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")

            # Open save dialog - FIXED: Removed problematic parameters
            file_path = filedialog.asksaveasfilename(
                title="Save output file",
                defaultextension=".txt",
                filetypes=[
                    ("Text files", "*.txt"),
                    ("All files", "*.*")
                ]
            )

            # Check if user cancelled the dialog
            if not file_path:
                return
            # Ensure the directory exists
            directory = os.path.dirname(file_path)
            if directory and not os.path.exists(directory):
                os.makedirs(directory, exist_ok=True)

            # Write file with explicit encoding and error handling
            with open(file_path, 'w', encoding='utf-8', newline='') as f:
                f.write(output)

            # Verify the file was created and has content
            if os.path.exists(file_path) and os.path.getsize(file_path) > 0:
                self.show_status_message(f"File saved successfully to: {os.path.basename(file_path)}",
"success")
            else:
                raise Exception("File was created but appears to be empty")

        except PermissionError:
            error_msg = "Permission denied. Please choose a different location or run as administrator."
            messagebox.showerror("Permission Error", error_msg)
            self.show_status_message("Save failed: Permission denied", "error")

        except FileNotFoundError:
            error_msg = "The specified path was not found. Please check the file path."
            messagebox.showerror("Path Error", error_msg)
            self.show_status_message("Save failed: Path not found", "error")

        except OSError as e:
            error_msg = f"Operating system error: {str(e)}"
            messagebox.showerror("OS Error", error_msg)
            self.show_status_message(f"Save failed: {str(e)}", "error")

        except UnicodeEncodeError as e:
            error_msg = f"Text encoding error: {str(e)}"
            messagebox.showerror("Encoding Error", error_msg)
            self.show_status_message("Save failed: Text encoding error", "error")

        except Exception as e:
            error_msg = f"Unexpected error while saving: {str(e)}"
            messagebox.showerror("Save Error", error_msg)
            self.show_status_message(f"Save failed: {str(e)}", "error")

def main():
    root = tk.Tk()
    app = TranspositionGUI(root)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Outputs



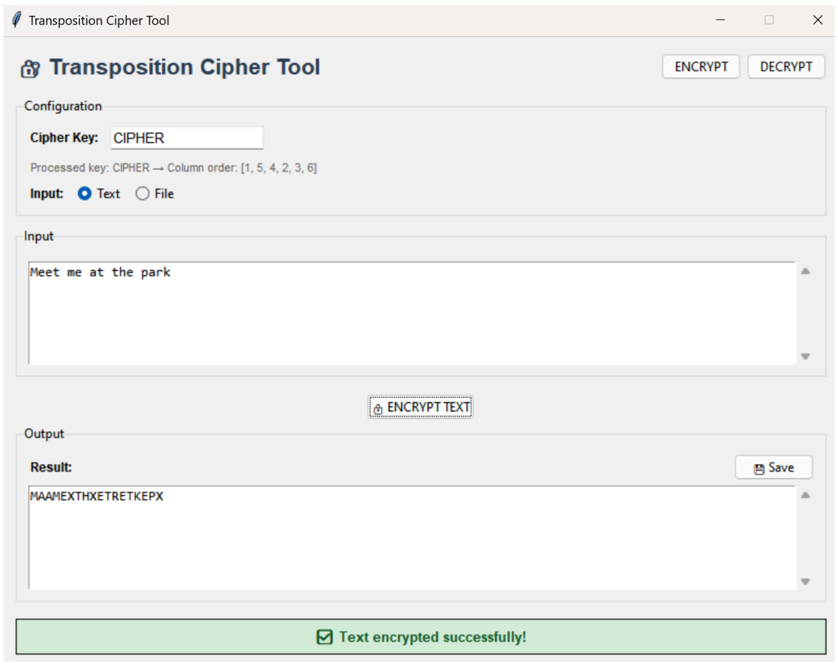
Cipher Key: CIPHER

Processed key: CIPHER → Column order: [1, 5, 4, 2, 3, 6]

Input: [✓ Text] File

Result:

Fig3. GUI Interface



Cipher Key: CIPHER

Processed key: CIPHER → Column order: [1, 5, 4, 2, 3, 6]

Input: [✓ Text] File

```
Meet me at the park
```

Result:

```
MAAMEXTHXETRETKEPX
```

Text encrypted successfully!

Fig4. Encrypted data

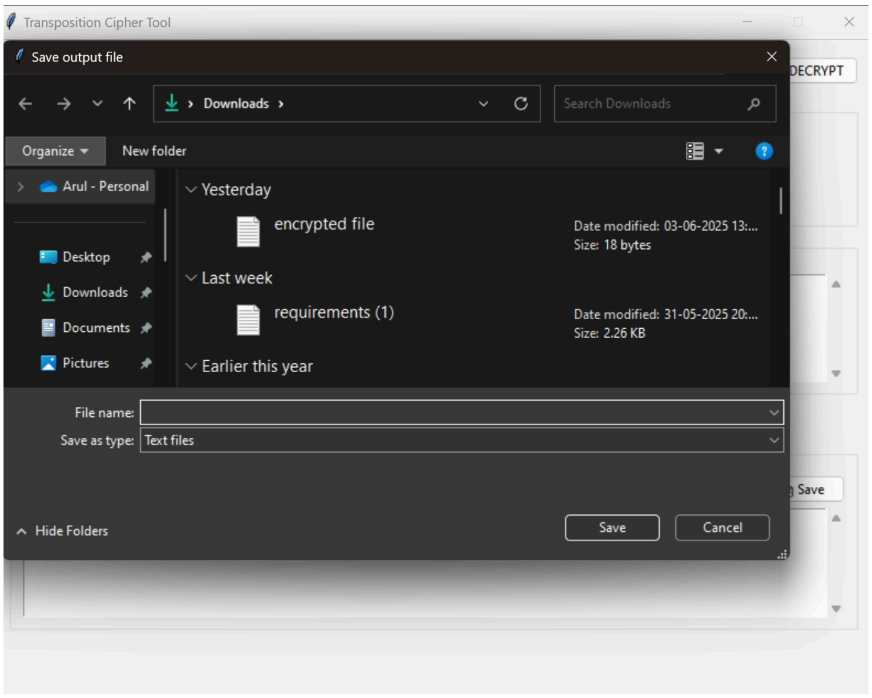
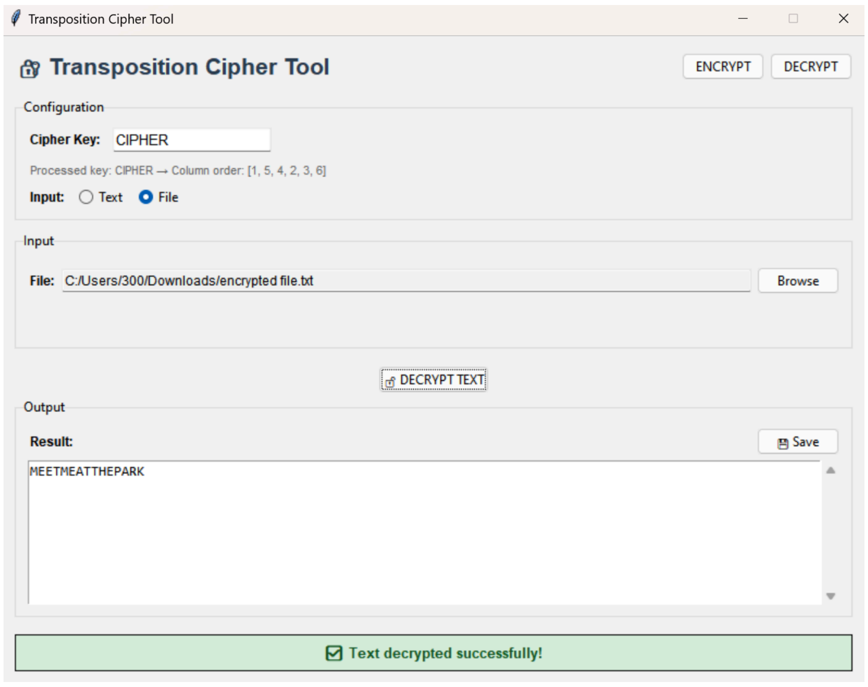


Fig5. Saving the output in a text file



Cipher Key: CIPHER

Processed key: CIPHER → Column order: [1, 5, 4, 2, 3, 6]

Input: ( ) Text [✓ File]

File: C:/Users/300/Downloads/encrypted file.txt

```
MEETMEATTHEPARK
```

Text decrypted successfully!

Fig6. Decryption Process

Conclusion

Through both theory and practice, this project demonstrates the principles of transposition cryptography. The implementation of a columnar transposition cipher not only illustrates how information can be scrambled to protect confidentiality, but also demonstrates how simple rearrangement methods can be combined with a key to create powerful encryption schemes. Furthermore, the development of a fully functional GUI with Python's Tkinter library enhances the usability of the tool which allows students and professionals to actively learn and understand cryptographic processes.

Aside from that, this project combines classic cryptographic theory and its implementation in modern programming. Modularizing the code into distinct classes for the cipher logic and GUI style interaction supports and provides a clean structure and easily scalable architecture which can be further built upon for more complex cryptographic methods. While achieving this goal, the project also teaches users how to work with transposition ciphers on a fundamental level, creating pathways for learning about broader contexts in computers, security, and data protection.

Although reliant on classical concepts, “columnar transposition” reinforces fundamental cryptographic techniques, such as confusion, key, diffusion, and key management. These practices are as important today as they were in ancient cryptosystems, thus building the understanding through active learning enables understanding of more advanced techniques used in modern cybersecurity encryption systems.

References

[1] D. Kahn, *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*, New York, NY, USA: Scribner, 1996.

[2] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed., Pearson, 2017.

[3] C. E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[4] S. Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Anchor Books, 1999.

[5] J. Buchmann, *Introduction to Cryptography*, 2nd ed., New York, NY, USA: Springer, 2004.

[6] G. C. Kessler, “An Overview of Cryptography,” [Online]. Available: https://www.garykessler.net/library/crypto.html.

[7] B. A. Forouzan and D. Mukhopadhyay, *Cryptography and Network Security*, New York, NY, USA: McGraw-Hill, 2007.

[8] A. Mardon, G. Barara, I. Chana, A. Di Martino, and I. Falade, "Cryptography," [Online]. Available: https://www.academia.edu/download/70204417/Cryptography_Online.pdf, 2021.

[9] N. U. Ezeonyi and O. R. Okonkwo, "Applications of Information Cryptography in Its Various Stages of Evolution, from Antiquity to the Modern Era," *JOMEEPS*, 2023. [Online]. Available: https://www.nigerianjournalsonline.com/index.php/JOMEEPS/article/view/3809

[10] J. Levinsky, "Encryption: The History and Implementation," SUNY, 2022. [Online]. Available: https://soar.suny.edu/handle/20.500.12648/12073

[11] M. J. Banks, "A Search-Based Tool for the Automated Cryptanalysis of Classical Ciphers," 2008. [Online]. Available: http://zodiacrevisited.com/wp-content/uploads/2015/04/mjb503_report.pdf

[12] M. Fransson, "Power Analysis of the Advanced Encryption Standard: Attacks and Countermeasures for 8-Bit Microcontrollers," 2015. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2:874463

[13] S. Hammoud, "Multiple Image Encryption for Business Applications using DNA Coding, Jigsaw Transform, and Chaos Theory," 2024. [Online]. Available: https://www.researchgate.net/publication/381126808

[14] O. S. Ajala, "Design and Implementation of an Improved Electronic Document Management System (Encodoc)," 2015. [Online]. Available: https://www.academia.edu/download/54404741/DESIGN_AND_IMPLEMENTATION_OF_AN_IMPROVED_ELECTRONIC_DOCUMENT_MANAGEMENT_SYSTEM_ENCODOC.pdf

[15] V. Rajasekar and J. Premalatha, “Introduction to classical cryptography,” in *Classical and Modern Cryptography*, Wiley, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119836728.ch1

[16] G. Lasry, N. Kopal, and A. Wacker, “Cryptanalysis of columnar transposition cipher with long keys,” *Cryptologia*, vol. 39, no. 4, pp. 315–335, 2016. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01611194.2015.1087074

[17] D. Arboledas Brihuega, “A new character-level encryption algorithm: How to implement cryptography in an ICT classroom,” *Journal of Technology and Science Education*, vol. 9, no. 1, 2019. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/172083/491-3193-1-PB.pdf

[18] OpenAI, *ChatGPT* (Jun. 1, 2025). Accessed on: Jun. 1, 2025. [Online]. Available: https://chat.openai.com/

[19] A. Armah, S. Asare, and E. Abrefah-Mensah, “Enhancing security in modern transposition ciphers through algorithmic innovations and advanced cryptanalysis,” *International Journal of Computer Science*, vol. 13, no. 3, 2024. [Online]. Available: http://ijcs.net/ijcs/index.php/ijcs/article/view/4095

[20] M. I. Bhat and K. J. Giri, “Impact of computational power on cryptography,” in *Advances in Information and Communication*, Springer, 2021. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-15-8711-5_4

[21] C. Swenson, *Modern Cryptanalysis: Techniques for Advanced Code Breaking*, Wiley, 2008. [Online]. Available: https://books.google.com/books?id=oLoaWgdmFJ8C

[22] J. O. Philip, “Development and statistical analysis of a hybrid classical encryption algorithm,” 2023. [Online]. Available: https://www.researchgate.net/publication/386454541

[23] A. Ketha, “The Evolution of Cryptography and a Contextual Analysis of the Major Modern Schemes,” *NHSJS*, 2023. [Online]. Available: https://nhsjs.com/wp-content/uploads/2024/04/The-Evolution-of-Cryptography-and-a-Contextual-Analysis-of-the-Major-Modern-Schemes.pdf

[24] Python Software Foundation, “tkinter — Python interface to Tcl/Tk,” *Python 3 Documentation*, 2024. [Online]. Available: https://docs.python.org/3/library/tkinter.html

[25] Python Software Foundation, “math — Mathematical functions,” *Python 3 Documentation*, 2024. [Online]. Available: https://docs.python.org/3/library/math.html

[26] Python Software Foundation, “os — Miscellaneous operating system interfaces,” *Python 3 Documentation*, 2024. [Online]. Available: https://docs.python.org/3/library/os.html

[27] Python Software Foundation, “datetime — Basic date and time types,” *Python 3 Documentation*, 2024. [Online]. Available: https://docs.python.org/3/library/datetime.html