**Declaration**

I undersigned solemnly declare that the project is based on my own work carried out during the course of my study. I assert that statement made and conclusions drawn are an outcome of my research work. I further certify that:

* The work contained in the report is original and has been done by me under the supervision of the academy.
* We have followed the guidelines provided by the university in writing the report.
* Whenever we have used material from other sources, we have given due credit to them in the text of the report and giving their details in reference.

Chitraksh Mahajan

**Acknowledgement**

To list who all have helped me in difficult because they are so numerous and depth is so enormous. I would like to acknowledge the following as idealistic channels and fresh dimension in the completion of this project.

I would like to express my sincere gratitude towards my project supervisor whose guidance and care made the project successful. We would like to express our sincere thanks to my supervisor.

I would also like to express gratitude towards SRM University for giving me this opportunity to do a project on **“CipherCraft: Fun with Ciphers”**. Without their support and suggestions, would not have been completed. Your valuable guidance and suggestions helped me in various phases of the completion of the project. I will always thankful to you in this regard.

I would like to thank college resources, for having provided various reference books and magazines related to our project. Lastly, I would like to thank each person who is directly or indirectly helped us in the completion of the project especially our parents and peer who supported me throughout my project.

I am ensuring that this project was finished by me and not copied.

**Abstract**

Cybersecurity shields our digital world from attacks. It's like a fortress, defending our online data, systems, and networks from hackers and malware. Encryption and firewalls are its tools, keeping information safe and private. But it's not just about technology; awareness and education play key roles too. Cyber threats keep evolving, demanding constant vigilance. Without cybersecurity, our online lives would be vulnerable to theft, fraud, and disruption. In today's interconnected world, where everything from banking to healthcare relies on digital systems, cybersecurity is essential for safeguarding our privacy, our finances, and even our safety.

Cryptography is the secret language of the digital age, securing our information in an increasingly interconnected world. It's the art and science of encoding and decoding messages, ensuring that only authorized parties can access sensitive data. From ancient times to modern-day digital communications, cryptography has evolved into a sophisticated field encompassing algorithms, keys, and protocols. Whether it's protecting financial transactions, securing military communications, or safeguarding personal privacy, cryptography plays a vital role. As technology advances, so too does the need for robust cryptographic systems to thwart malicious actors. In essence, cryptography is the guardian of our digital secrets, enabling secure communication and trust in the digital realm.

Encryption shields data with intricate algorithms, rendering it indecipherable without the correct decryption key. Crucial in cybersecurity, it ensures data confidentiality, integrity, and authenticity, thwarting unauthorized access and tampering. From ancient methods like Caesar cipher to modern symmetric and asymmetric encryption, it evolves to counter digital threats. Encryption's application spans secure communication channels to protecting sensitive data, preserving privacy, and securing digital assets amidst an interconnected landscape fraught with cyber risks.

**Table of Content**

1. **Declaration………………………………………………………...…….1**
2. **Acknowledgement…………………………………………………........2**
3. **Abstract………………………………………………………………….3**
4. **Introduction.…………………………………….……………………….5**
   1. Objectives…………………………………………………………………………..5
5. **Literature.......................................................................................................5-9**
   1. Cryptography……………………………………………………………….5
   2. Types of Cryptography……………………………………………………..6
   3. Strengths and Weaknesses of Cryptography……………………………….7
   4. Government Access to Keys………………………………………………...8
   5. Ciphers……………………………………………………………………...8
   6. Types of Ciphers…………………………………………………………....8-9
   7. Importance of Ciphers in Modern Computing……………………………...9
6. **Technology Stack…………………………………………………...……..10-13**
   1. Programming Language used………………………………………………..10
   2. Libraries and modules used………………………………………………….10-12
   3. Development Tools…………………………………………………………..12-13
7. **Ciphers Implemented………………………………………………….….13-18**
   1. Caesar Cipher………………………………………………………………...13-14
   2. Atbash Cipher………………………………………………………………...14-15
   3. Rail Fence Cipher…………………………………………………………….16-17
   4. Keyword Cipher……………………………………………………………...17-18
8. **User Interface………………………………………………………..…….19-22**
   1. Screenshots……………….…………………………………………………...19-21
   2. User Interaction Flow………………………………………………………....21-22
9. **Program Code………………………………………………………….….23-40**
10. **Conclusion…………………………………………………………………40-41**
    1. Summary of Findings………………………………………………………....40-41
    2. Lesson Learned……………………………………………………………......41
11. **References………………………………………………………………......42**

1. **Introduction**

In an era marked by digital communication and the pervasive exchange of sensitive information, ensuring the security and confidentiality of messages has become paramount. CipherCraft emerges as a solution, poised at the intersection of encryption technology and user-centric design. With a fundamental objective of safeguarding digital communications, CipherCraft offers users a seamless platform for encrypting and decrypting messages using a variety of cryptographic ciphers.

* 1. **Objective**

The primary objective of developing CipherCraft is to provide users with a comprehensive tool for securing their messages through encryption while ensuring ease of use and accessibility. By leveraging CipherCraft, users can encrypt their messages using robust cryptographic algorithms, customize cipher settings to suit their needs, and decrypt messages with confidence, knowing that their communications remain private and secure.

Through this project, we aim to introduce CipherCraft to our target audience, delineate its purpose, and outline its key functionalities. By developing CipherCraft, we strive to empower users with a user-friendly yet powerful tool for safeguarding their digital communications, thereby enhancing their privacy and security in an increasingly interconnected world.

1. **Literature**
   1. **Cryptography**

Cryptography, originating from the Greek words "**kryptos**" (hidden) and "**graphein**" (to write), is the science and practice of secure communication in the presence of adversaries. It involves techniques for encrypting and decrypting information to ensure confidentiality, integrity, and authenticity.

Cryptography revolves around concepts like encryption and decryption, where plaintext (readable data) is transformed into ciphertext (unreadable data) using algorithms and secret keys. These algorithms include symmetric key algorithms like AES and asymmetric key algorithms like RSA. Key management is crucial in cryptography, involving the secure generation, distribution, storage, and revocation of cryptographic keys. Additionally, cryptographic hash functions play a role in data integrity verification, digital signatures, and password hashing.

Applications of cryptography include data confidentiality, integrity, authentication, and non-repudiation. Its principles and applications are fundamental in modern cybersecurity, enabling secure communication, data protection, and identity verification.

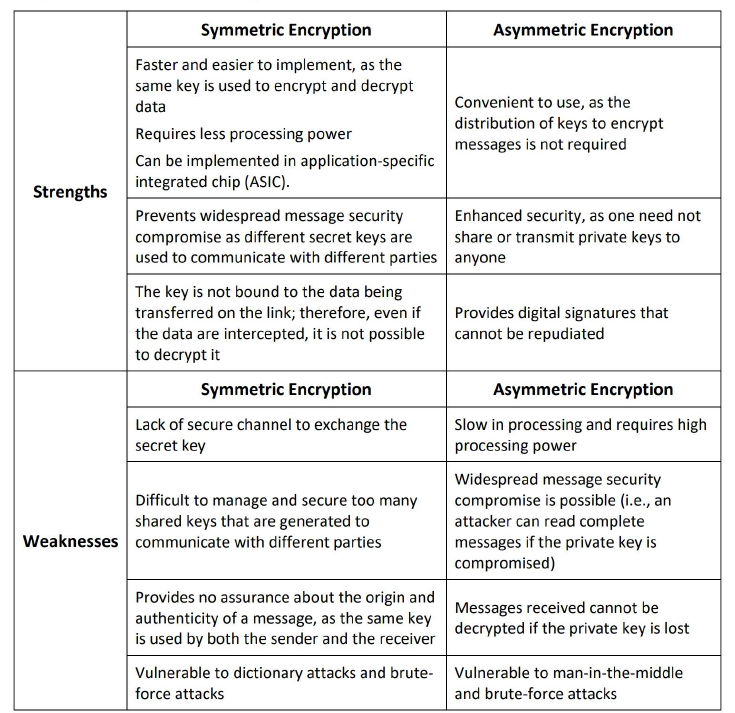
* 1. **Types of Cryptography**

Cryptography can be broadly categorized into two main types: symmetric cryptography and asymmetric cryptography.

**1. Symmetric Cryptography:**

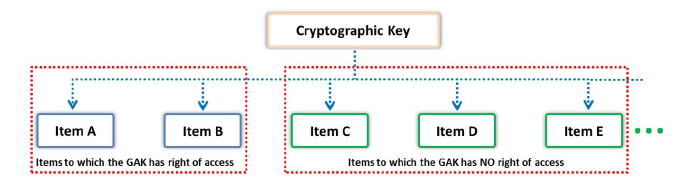
* In symmetric cryptography, the same key is used for both encryption and decryption. This means that the sender and receiver must both possess the same secret key. It is typically faster and more efficient than asymmetric cryptography, making it suitable for encrypting large amounts of data.
* However, a major challenge with symmetric cryptography is key distribution. Since the same key is used for encryption and decryption, securely sharing the key between communicating parties can be difficult.
* Common symmetric encryption algorithms include Advanced Encryption Standard (AES), Data Encryption Standard (DES), and Triple DES (3DES).

**2. Asymmetric Cryptography:**

* Asymmetric cryptography, also known as public-key cryptography, uses a pair of keys: a public key and a private key. The public key is freely distributed, while the private key is kept secret. Data encrypted with the public key can only be decrypted with the corresponding private key, and vice versa. It is typically slower and more computationally intensive than symmetric cryptography.
* Asymmetric cryptography addresses the key distribution problem inherent in symmetric cryptography by allowing parties to communicate securely without sharing secret keys.
* Common asymmetric encryption algorithms include Rivest-Shamir-Adleman (RSA), Elliptic Curve Cryptography (ECC), and Digital Signature Algorithm (DSA).
  1. **Strengths and Weaknesses of Cryptography**
  2. **Government Access to Keys**

Government access to encryption keys refers to the ability of government authorities to gain access to encrypted data, typically for law enforcement or national security purposes. Encryption is a crucial technology that ensures the confidentiality and integrity of digital communications by scrambling the content in such a way that only authorized parties can decipher it.

However, this encryption can pose challenges for law enforcement agencies, especially when investigating criminal activities or combating terrorism. Advocates for government access argue that it is necessary for maintaining public safety and national security by enabling authorities to intercept and decrypt communications relevant to ongoing investigations.

They suggest that lawful access mechanisms, such as encryption backdoors or legal frameworks for accessing encryption keys, are essential tools for combating crime in the digital age.

* 1. **Ciphers**

Ciphers are cryptographic algorithms used to encrypt and decrypt messages or data. They operate by substituting or transposing characters according to a specific rule or key, making the original message unintelligible to anyone without the key. There are two main types of ciphers: symmetric and asymmetric.

Ciphers play a crucial role in ensuring the confidentiality, integrity, and authenticity of digital communications and transactions, forming the foundation of modern cybersecurity protocols and systems.

* 1. **Types of Ciphers**

1. **Classical Ciphers**

These are historical encryption techniques predating the modern computer era. Classical ciphers often rely on simple substitution and transposition techniques.

Examples include the Caesar cipher, Atbash cipher, and Vigenère cipher.

1. **Modern Ciphers**
   1. Based on Key used:
      * Symmetric Key Ciphers: Also known as secret key ciphers, these use a single key for both encryption and decryption. Examples include Data Encryption Standard (DES), Advanced Encryption Standard (AES), and Blowfish.
      * Asymmetric Key Ciphers: Also called public-key ciphers, they use a pair of keys: a public key for encryption and a private key for decryption. Examples include RSA (Rivest-Shamir-Adleman) and Elliptic Curve Cryptography (ECC).
   2. Based on Input Data Type:
      * Stream Ciphers: These encrypt plaintext bit by bit or byte by byte, typically used for real-time communication. They are efficient for encrypting large data streams. Examples include RC4 and Salsa20.
      * Block Ciphers: These encrypt fixed-size blocks of data at a time, commonly used in data storage and transmission. They require padding for variable-length data. Examples include AES and DES.
   3. **Importance of Ciphers in Modern Computing**

Ciphers play a crucial role in modern computing by ensuring the security and confidentiality of digital data. Their importance stems from several key factors:

1. **Data Confidentiality:** Ciphers encrypt sensitive information, such as personal data, financial transactions, and corporate communications, preventing unauthorized access and protecting privacy.
2. **Secure Communication:** Ciphers enable secure communication over networks by encrypting data transmitted between parties. This ensures that only authorized recipients can access the information, safeguarding it from interception or eavesdropping.
3. **Data Integrity:** Ciphers not only encrypt data but also provide mechanisms for verifying its integrity. Hash functions, often used in conjunction with encryption, generate fixed-size hash values (digests) for input data. These hashes can detect any tampering or unauthorized modifications to the encrypted data.
4. **Compliance and Regulation:** Many industries and organizations are subject to regulatory requirements regarding data protection and privacy. Encryption, facilitated by ciphers, helps these entities comply with regulations such as the GDPR (General Data Protection Regulation) and HIPAA (Health Insurance Portability and Accountability Act).
5. **Protection Against Cyber Threats:** In an era of increasing cyber threats, including data breaches, ransomware attacks, and identity theft, encryption provided by ciphers serves as a critical defense mechanism. It mitigates the risk of data exposure and financial losses resulting from unauthorized access to sensitive information.
6. **Technology Stack**

**3.1. Programming Languages used**

**Python:**

It is a versatile and widely-used programming language, forms the foundation of CipherCraft's development. Renowned for its simplicity, readability, and ease of use, Python offers numerous advantages for software development projects like CipherCraft.

One of Python's key strengths lies in its extensive standard library, which provides a wide array of modules and packages for various tasks, including encryption, data manipulation, and user interface design. For CipherCraft, Python's standard library likely facilitated the implementation of cryptographic algorithms and user interface components.

Moreover, Python's dynamic typing and high-level syntax contribute to faster development cycles and easier debugging, enabling developers to focus more on implementing features rather than managing low-level details. This aspect of Python's design likely streamlined the development process of CipherCraft, allowing for rapid prototyping and iterative improvements.

Additionally, Python's active and supportive community ensures access to a wealth of resources, tutorials, and third-party libraries that can enhance CipherCraft's functionality. Whether integrating with external APIs for key management or implementing advanced encryption techniques, Python's ecosystem likely provided ample resources to support CipherCraft's development needs.

Furthermore, Python's cross-platform compatibility ensures that CipherCraft can be deployed and used on various operating systems without significant modifications, enhancing its accessibility and usability for a wider audience.

Overall, Python's simplicity, versatility, and robust ecosystem make it an ideal choice for developing applications like CipherCraft, where security, usability, and efficiency are paramount.

**3.2. Libraries and modules used**

* **Tkinter:**

CipherCraft relies exclusively on the Tkinter library for its graphical user interface (GUI) development. Tkinter, a standard GUI toolkit for Python, offers a simple yet powerful way to create interactive applications with ease.

By leveraging Tkinter, CipherCraft benefits from its integration with the Tk GUI toolkit, providing access to a wide range of GUI components such as buttons, labels, text entry fields, and more. These components enable the creation of a user-friendly interface for tasks such as message input, cipher selection, and encryption/decryption controls.

Tkinter's cross-platform compatibility ensures that CipherCraft's GUI remains consistent and functional across different operating systems, including Windows, macOS, and Linux. This characteristic enhances the accessibility of CipherCraft, allowing users on various platforms to utilize its encryption functionalities seamlessly. Its intuitive event-driven programming model facilitates the handling of user interactions, ensuring smooth navigation and operation within the application.

Additionally, Tkinter's extensive documentation and online resources provide developers with valuable guidance and support throughout the development process. Whether designing layout structures, handling user input, or customizing widget appearance, Tkinter offers comprehensive documentation and community forums to address developers' needs effectively.

Overall, Tkinter serves as the cornerstone of CipherCraft's GUI development, offering a robust framework for creating interactive interfaces that enhance user engagement and usability.

* **PyperClip:**

CipherCraft utilizes the pyperclip module to facilitate clipboard operations within the application. pyperclip provides a simple cross-platform Python module for copying and pasting text to and from the clipboard, making it an essential component for CipherCraft's functionality.

By incorporating pyperclip, CipherCraft enables users to conveniently copy encrypted or decrypted messages to the system clipboard for easy sharing or pasting into other applications. Likewise, users can paste text from the clipboard directly into CipherCraft for encryption or decryption, streamlining the input process.

Pyperclip abstracts the differences in clipboard handling across various operating systems, ensuring consistent behavior and compatibility with Windows, macOS, and Linux environments. This compatibility enhances the usability of CipherCraft across different platforms, allowing users to perform clipboard operations seamlessly regardless of their operating system.

Furthermore, pyperclip's straightforward API simplifies integration into CipherCraft's codebase, enabling developers to implement clipboard functionality with minimal effort. Its intuitive interface for copying and pasting text abstracts the underlying platform-specific details, providing a user-friendly experience within CipherCraft.

Overall, the pyperclip module plays a vital role in enhancing CipherCraft's usability by facilitating clipboard operations, thereby improving the efficiency of text input and output within the application.

**3.3. Development Tools**

* **Visual Studio Code:**

Visual Studio Code (VS Code) is a lightweight and versatile source code editor developed by Microsoft for Windows, macOS, and Linux. It provides a rich set of features for various programming languages, including Python, making it a popular choice among developers for software development projects.

In the development of CipherCraft, VS Code served as the primary Integrated Development Environment (IDE) due to its ease of use, extensive functionality, and support for Python programming. Developers utilized VS Code for writing, editing, and organizing the project's Python codebase. Its intuitive interface and powerful features, such as syntax highlighting, auto-completion, and code debugging, streamlined the coding process and enhanced productivity.

Overall, VS Code provided a robust and efficient development environment for the CipherCraft project, enabling developers to write, test, and maintain the application code with ease and efficiency.

* **ChatGPT:**

ChatGPT, a state-of-the-art language model developed by OpenAI, played a crucial role in the development of CipherCraft. As a powerful AI language model, ChatGPT is capable of understanding and generating human-like text based on the input it receives. In the context of CipherCraft's development,

ChatGPT served as a virtual assistant and a knowledge repository for the developers. Developers leveraged ChatGPT to explore cryptographic concepts, clarify programming doubts, and brainstorm ideas for implementing various features of CipherCraft. By engaging in conversational interactions with ChatGPT, developers could quickly access information about different encryption algorithms, understand the intricacies of cryptography, and receive guidance on Python programming techniques relevant to the project.

Furthermore, ChatGPT provided valuable assistance in code refinement. Developers could input snippets of Python code or describe the functionality they desired, and ChatGPT would offer suggestions, corrections, or alternative approaches to improve the code's efficiency and readability. This collaborative process streamlined the development workflow, enabling developers to iterate rapidly and achieve better outcomes in building CipherCraft.

In summary, ChatGPT's versatility and proficiency in natural language understanding significantly contributed to the development of CipherCraft by serving as an intelligent assistant, knowledge resource, and code optimization tool for the development team.

1. **Ciphers Implemented**

**4.1. Caesar Cipher**

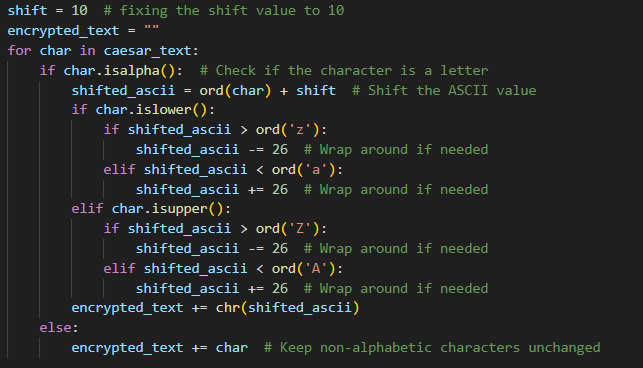
Caesar cipher, named after **Julius Caesar** who reportedly used it for secure communication, is one of the simplest and earliest known encryption techniques. It is a substitution cipher where each letter in the plaintext is shifted a certain number of places down or up the alphabet. For example, with a shift of 3, A would be replaced by D, B would become E, and so on.

In CipherCraft, the Caesar cipher is implemented as a fundamental encryption method. Users can choose a specific shift value to encrypt their messages. This implementation allows for quick and easy encryption, making it suitable for basic encryption needs.

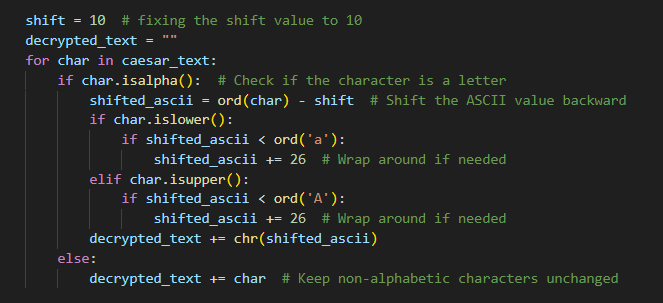
However, despite its historical significance and simplicity, the Caesar cipher is highly vulnerable to brute force attacks and frequency analysis due to its limited key space (only 25 possible shifts in the English alphabet). Therefore, it is not suitable for encrypting sensitive information but serves as an educational tool for understanding basic encryption principles.

In CipherCraft, the Caesar cipher serves as a foundational component for users to grasp the concept of encryption and explore the functionality of the application. It provides a hands-on experience for users to encrypt and decrypt messages using a straightforward algorithm, laying the groundwork for understanding more complex ciphers implemented in the application.

Encryption Code:



Decryption Code:



**4.2. Atbash Cipher**

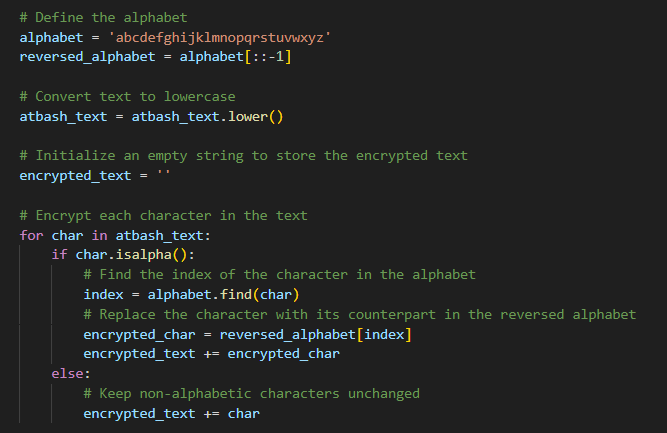
The Atbash cipher is a monoalphabetic substitution cipher that operates by replacing each letter in the plaintext with its reverse counterpart in the alphabet. For instance, in the English alphabet, A becomes Z, B becomes Y, and so forth. It derives its name from the first and last letters of the Hebrew alphabet, Aleph and Tav, symbolizing the reversal process.

In CipherCraft, the Atbash cipher is another encryption method available to users. It offers a simple yet effective way to obfuscate plaintext messages. Users can input their messages, and CipherCraft automatically encrypts them using the Atbash algorithm. This cipher provides a basic level of encryption suitable for casual communication or educational purposes.

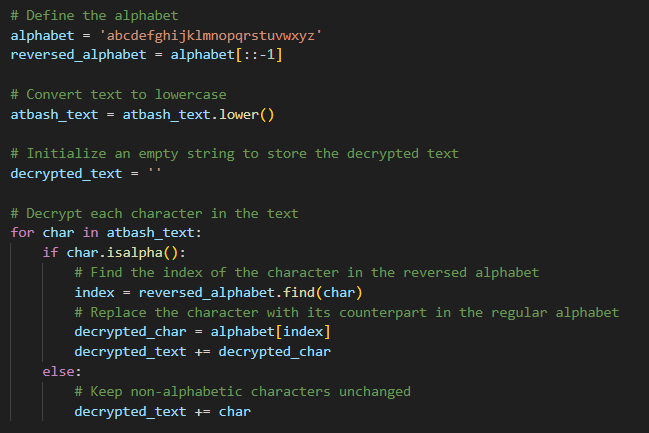
Despite its simplicity, the Atbash cipher can obscure the meaning of messages effectively, making it useful in contexts where a basic level of confidentiality is desired. However, similar to other monoalphabetic substitution ciphers, it is vulnerable to frequency analysis and other cryptanalysis techniques, limiting its effectiveness for securing sensitive information.

In CipherCraft, the Atbash cipher contributes to the application's educational value by demonstrating various methods of transforming plaintext into ciphertext, enriching users' learning experience in cryptography.

Encryption Code:



Decryption Code:



**4.3. Rail Fence Cipher**

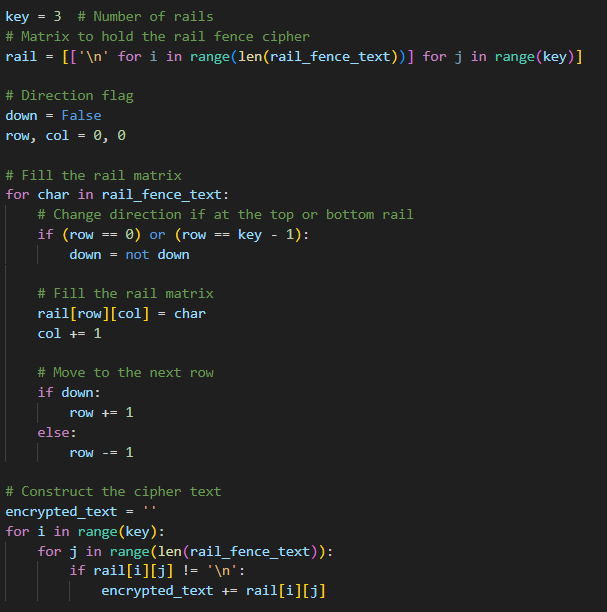
The Rail Fence cipher, also known as zigzag cipher, is a transposition cipher that rearranges the plaintext characters in a zigzag pattern along an imaginary rail fence. It works by writing the plaintext in a diagonal manner across a specified number of "rails" or lines, and then reading off the ciphertext by following the rail pattern.

The Rail Fence cipher is implemented as another encryption option. The plaintext message is then written diagonally across 3 rails, and CipherCraft generates the ciphertext by reading off the characters in a zigzag pattern.

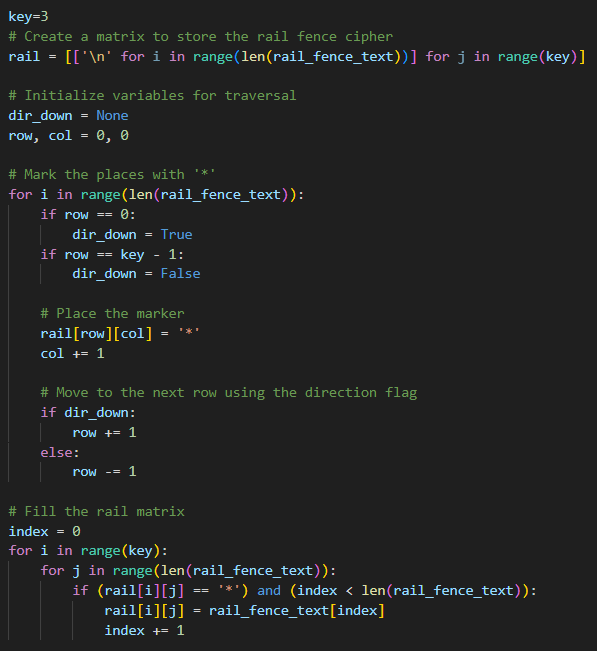
This cipher provides a basic yet intriguing method of encryption, suitable for educational purposes or casual communication where a moderate level of security is sufficient. However, it is susceptible to cryptanalysis techniques such as brute force attacks, especially for smaller rail counts.

By experimenting with different rail counts, users can observe how the complexity of the encryption process varies and appreciate the importance of key management in cryptographic systems.

Encryption Code:



Decryption Code:



**4.4. Keyword Cipher**

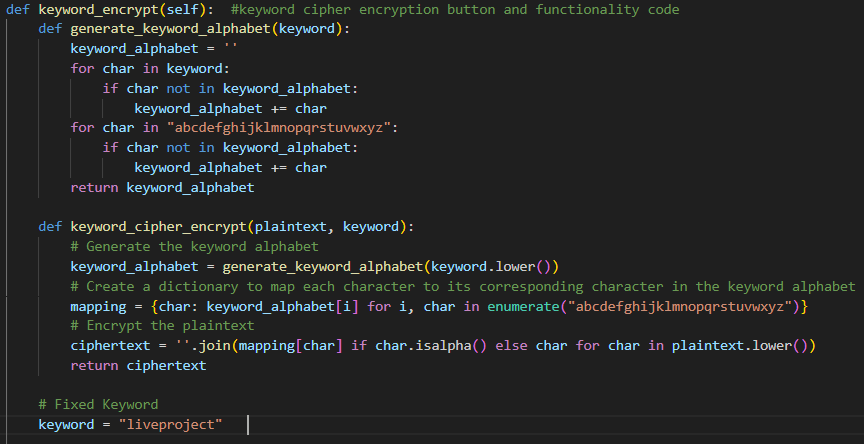
The Keyword Cipher, also known as the Keyword Shift cipher, is a substitution cipher that involves shifting the letters of the alphabet based on a keyword. It is a polyalphabetic substitution cipher, meaning that different parts of the message may be encoded using different alphabets.

In CipherCraft, the Keyword Cipher is implemented as an encryption option. Users can specify a keyword, which determines the shifting of letters in the alphabet for encryption. The letters of the keyword are appended to the beginning of the alphabet, and any duplicates are removed to create the cipher alphabet. The plaintext message is then encrypted by substituting each letter with its corresponding letter in the cipher alphabet.

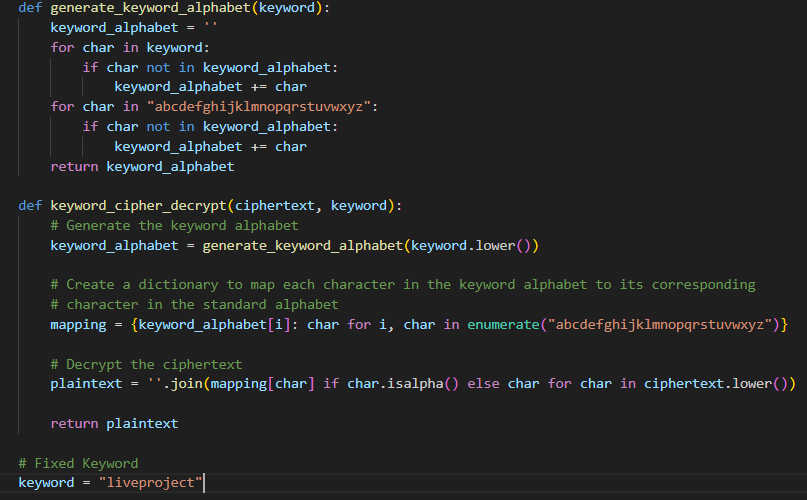
The Keyword Cipher is susceptible to frequency analysis and other cryptanalysis techniques, particularly if the keyword is short or easily guessable.

Despite its vulnerability, the Keyword Cipher offers an engaging way for users to explore the concept of substitution ciphers and understand the importance of key selection in cryptography. By experimenting with different keywords and observing the resulting ciphertext, users can gain insights into the strengths and limitations of polyalphabetic substitution techniques.

Encryption Code:



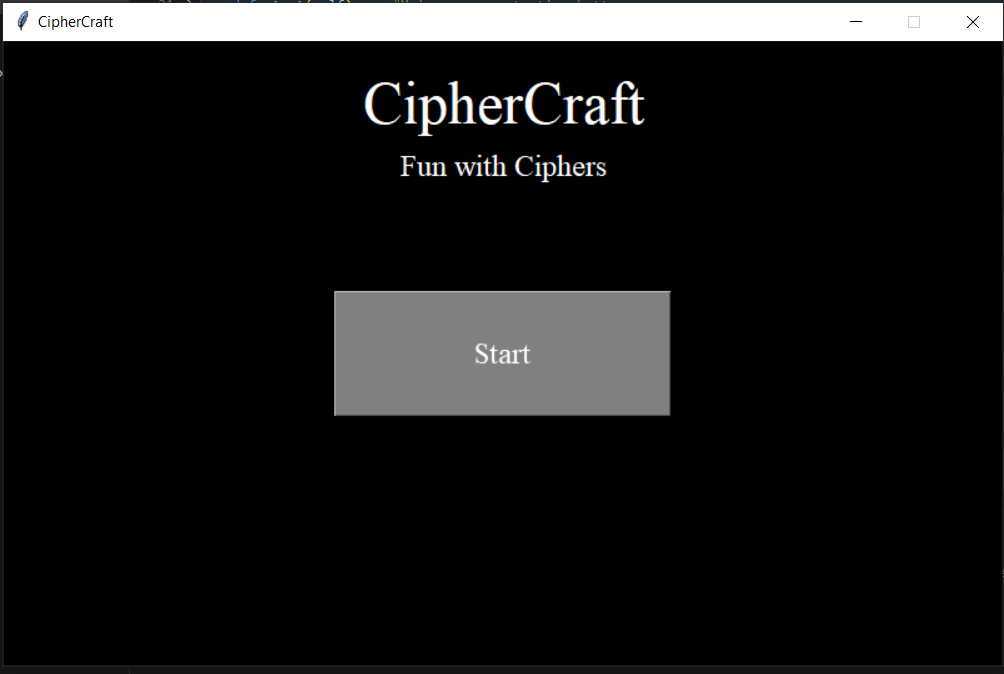
Decryption Code:



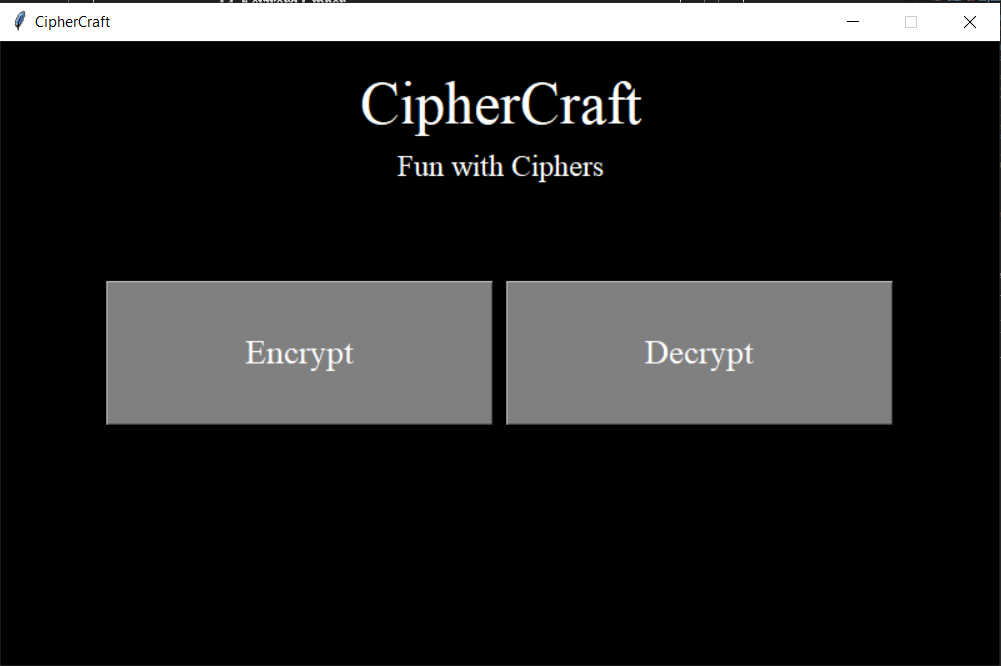
1. **User Interface**

**5.1. Screenshots**

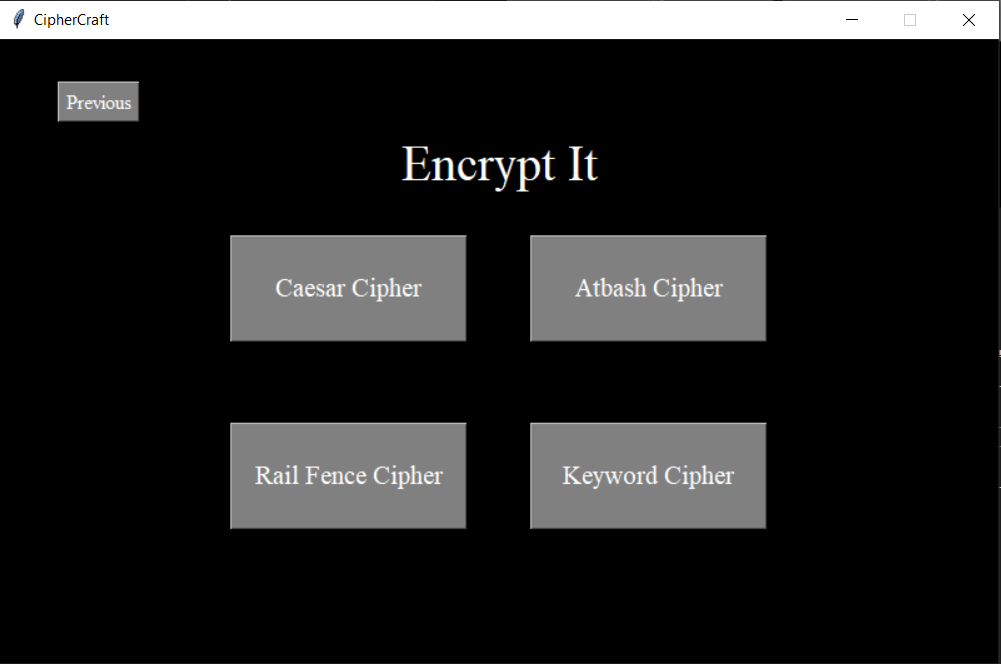
Program Start Screen



Program Main Screen



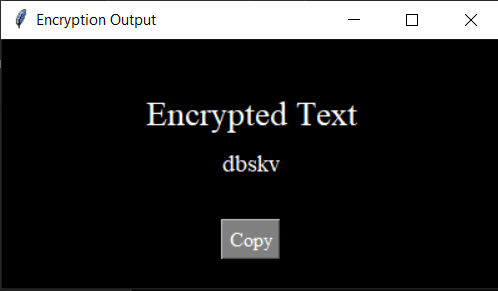
“Encrypt” screen



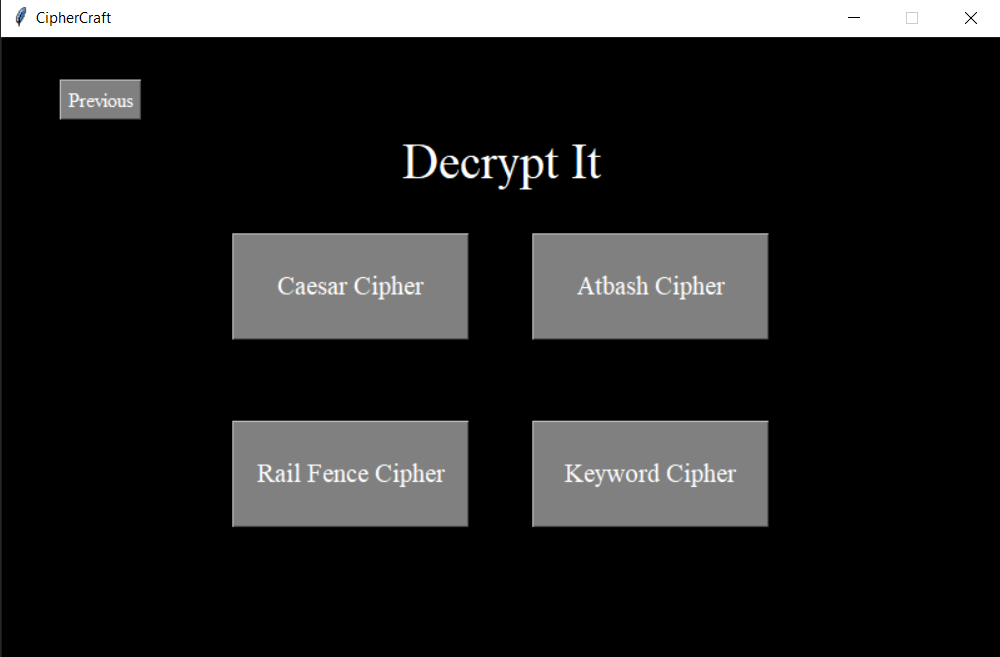
“Caesar Cipher” Screen



Final “Encrypt” Screen



“Decrypt” screen

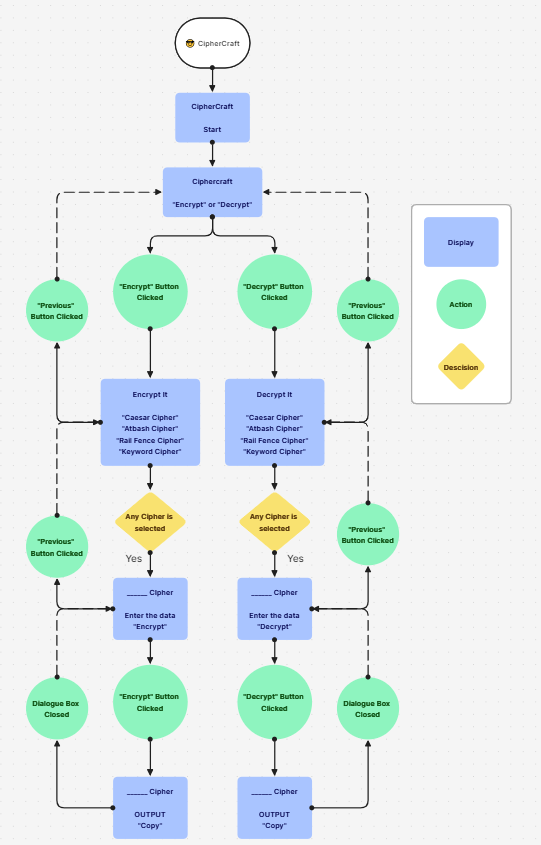


“Caesar Cipher” Screen



Final “Decrypt” Screen



**5.2. User Interaction Flow**

1. **Program Code**
2. import tkinter as tk
3. import pyperclip
4. class CipherCraftApp:
5. def \_\_init\_\_(self, root): # Program starting Landing screen
6. self.root = root
7. self.root.title("CipherCraft")
8. self.root.geometry("800x500")#box size
9. self.root.config(bg="black")  # Set background color
10. self.root.resizable(False, False)  # Disable window resizing
12. # Logo and slogan
13. self.logo\_label = tk.Label(root, text="CipherCraft", font=("Times New Roman", 36), fg="white", bg="black")
14. self.logo\_label.place(relx=0.5, rely=0.1, anchor="center")
15. self.slogan\_label = tk.Label(root, text="Fun with Ciphers", font=("Times New Roman", 18), fg="white", bg="black")
16. self.slogan\_label.place(relx=0.5, rely=0.2, anchor="center")
17. # Encrypt button
18. self.encrypt\_button = tk.Button(root, text="Start", command=self.start, width=20, height=3, font=("Times New Roman", 18), bg="gray", fg="white")
19. self.encrypt\_button.place(relx=0.5, rely=0.5, anchor="center")
21. def start(self):    #Main program starting button
22. # Clear the root window
23. self.clear\_screen()
25. # Logo and slogan
26. self.logo\_label = tk.Label(root, text="CipherCraft", font=("Times New Roman", 36), fg="white", bg="black")
27. self.logo\_label.place(relx=0.5, rely=0.1, anchor="center")
28. self.slogan\_label = tk.Label(root, text="Fun with Ciphers", font=("Times New Roman", 18), fg="white", bg="black")
29. self.slogan\_label.place(relx=0.5, rely=0.2, anchor="center")
31. # Encrypt button
32. self.encrypt\_button = tk.Button(root, text="Encrypt", command=self.cipher\_types\_encrypt, width=20, height=3, font=("Times New Roman", 20), bg="gray", fg="white")
33. self.encrypt\_button.place(relx=0.30, rely=0.5, anchor="center")
35. # Decrypt button
36. self.decrypt\_button = tk.Button(root, text="Decrypt", command=self.cipher\_types\_decrypt, width=20, height=3, font=("Times New Roman", 20), bg="gray", fg="white")
37. self.decrypt\_button.place(relx=0.70, rely=0.5, anchor="center")

40. def cipher\_types\_encrypt(self): #Encrypt It screen
41. # Clear the root window
42. self.clear\_screen()
44. # New screen for encryption
45. encrypt\_label = tk.Label(self.root, text="Encrypt It", font=("Times New Roman", 30), fg="white", bg="black")
46. encrypt\_label.place(relx=0.5, rely=0.2, anchor="center")
48. # Caesar Cipher button
49. caesar\_button = tk.Button(self.root, text="Caesar Cipher", command=self.caesar\_cipher,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
50. caesar\_button.place(relx=0.35, rely=0.4, anchor="center")
51. # Atbash Cipher button
52. atbash\_button = tk.Button(self.root, text="Atbash Cipher", command=self.atbash\_cipher,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
53. atbash\_button.place(relx=0.65, rely=0.4, anchor="center")
55. # Rail Fence Cipher button
56. rail\_fence\_button = tk.Button(self.root, text="Rail Fence Cipher", command=self.rail\_fence\_cipher,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
57. rail\_fence\_button.place(relx=0.35, rely=0.7, anchor="center")
59. # keyword Cipher button
60. keyword\_button = tk.Button(self.root, text="Keyword Cipher", command=self.keyword\_cipher,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
61. keyword\_button.place(relx=0.65, rely=0.7, anchor="center")
63. # Previous button
64. previous\_button = tk.Button(self.root, text="Previous", command=self.start, font=("Times New Roman", 12), bg="gray", fg="white")
65. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
67. def caesar\_cipher(self):    # caesar cipher encrypt screen
68. # Clear the root window
69. self.clear\_screen()
71. # New screen for Caesar Cipher
72. caesar\_label = tk.Label(self.root, text="Caesar Cipher", font=("Times New Roman", 28), fg="white", bg="black")
73. caesar\_label.place(relx=0.5, rely=0.2, anchor="center")
75. # Input box label
76. input\_label = tk.Label(self.root, text="Input Text to be Encrypted:", font=("Times New Roman", 14), fg="white", bg="black")
77. input\_label.place(relx=0.3, rely=0.4, anchor="center")
78. # Input text box
79. self.input\_entry = tk.Text(self.root, width=40, height=5)
80. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
82. # Encrypt button
83. encrypt\_button = tk.Button(self.root, text="Encrypt", command=self.caesar\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
84. encrypt\_button.place(x=400,y=350, anchor="center")
86. # Previous button
87. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
88. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
89. def caesar\_encrypt(self):   #caesar cipher encryption button and funcitonality code
90. # Get text from the input entry in the text box
91. caesar\_text = self.input\_entry.get("1.0", "end-1c")
92. self.input\_entry.delete("1.0", "end")
94. shift = 10  # fixing the shift value to 10
95. encrypted\_text = ""
96. for char in caesar\_text:
97. if char.isalpha():  # Check if the character is a letter
98. shifted\_ascii = ord(char) + shift  # Shift the ASCII value
99. if char.islower():
100. if shifted\_ascii > ord('z'):
101. shifted\_ascii -= 26  # Wrap around if needed
102. elif shifted\_ascii < ord('a'):
103. shifted\_ascii += 26  # Wrap around if needed
104. elif char.isupper():
105. if shifted\_ascii > ord('Z'):
106. shifted\_ascii -= 26  # Wrap around if needed
107. elif shifted\_ascii < ord('A'):
108. shifted\_ascii += 26  # Wrap around if needed
109. encrypted\_text += chr(shifted\_ascii)
110. else:
111. encrypted\_text += char  # Keep non-alphabetic characters unchanged

114. # Open a new window to display output
115. output\_window = tk.Toplevel(self.root)
116. output\_window.title("Encryption Output")
117. output\_window.geometry("400x200")
118. output\_window.config(bg="black")
120. output\_label = tk.Label(output\_window, text="Encrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
121. output\_label.place(relx=0.5, rely=0.3, anchor="center")
123. output\_text = tk.Label(output\_window, text=encrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
124. output\_text.place(relx=0.5, rely=0.5, anchor="center")
126. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(encrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
127. copy\_button.place(relx=0.5, rely=0.8, anchor="center")
129. def atbash\_cipher(self):    # atbash cipher encrypt screen
130. # Clear the root window
131. self.clear\_screen()
133. # New screen for Substitutional Cipher
134. substitution\_label = tk.Label(self.root, text="Atbash Cipher", font=("Times New Roman", 28), fg="white", bg="black")
135. substitution\_label.place(relx=0.5, rely=0.2, anchor="center")
137. # Input box label
138. input\_label = tk.Label(self.root, text="Input Text to be Encrypted:", font=("Times New Roman", 14), fg="white", bg="black")
139. input\_label.place(relx=0.3, rely=0.4, anchor="center")
140. # Input text box
141. self.input\_entry = tk.Text(self.root, width=40, height=5)
142. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
144. # Encrypt button
145. encrypt\_button = tk.Button(self.root, text="Encrypt", command=self.atbash\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
146. encrypt\_button.place(x=400,y=350, anchor="center")
148. # Previous button
149. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
150. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
151. def atbash\_encrypt(self):   #atbash cipher encryption button and funcitonality code
152. # Get text from the input entry in the text box
153. atbash\_text = self.input\_entry.get("1.0", "end-1c")
154. self.input\_entry.delete("1.0", "end")
156. # Define the alphabet
157. alphabet = 'abcdefghijklmnopqrstuvwxyz'
158. reversed\_alphabet = alphabet[::-1]
160. # Convert text to lowercase
161. atbash\_text = atbash\_text.lower()
163. # Initialize an empty string to store the encrypted text
164. encrypted\_text = ''
166. # Encrypt each character in the text
167. for char in atbash\_text:
168. if char.isalpha():
169. # Find the index of the character in the alphabet
170. index = alphabet.find(char)
171. # Replace the character with its counterpart in the reversed alphabet
172. encrypted\_char = reversed\_alphabet[index]
173. encrypted\_text += encrypted\_char
174. else:
175. # Keep non-alphabetic characters unchanged
176. encrypted\_text += char
178. # Open a new window to display output
179. output\_window = tk.Toplevel(self.root)
180. output\_window.title("Encryption Output")
181. output\_window.geometry("400x200")
182. output\_window.config(bg="black")
184. output\_label = tk.Label(output\_window, text="Encrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
185. output\_label.place(relx=0.5, rely=0.3, anchor="center")
187. output\_text = tk.Label(output\_window, text=encrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
188. output\_text.place(relx=0.5, rely=0.5, anchor="center")
190. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(encrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
191. copy\_button.place(relx=0.5, rely=0.8, anchor="center")
193. def rail\_fence\_cipher(self):    # rail fence cipher encrypt screen
194. # Clear the root window
195. self.clear\_screen()
197. # New screen for Substitutional Cipher
198. substitution\_label = tk.Label(self.root, text="Rail Fence Cipher", font=("Times New Roman", 28), fg="white", bg="black")
199. substitution\_label.place(relx=0.5, rely=0.2, anchor="center")
201. # Input box label
202. input\_label = tk.Label(self.root, text="Input Text to be Encrypted:", font=("Times New Roman", 14), fg="white", bg="black")
203. input\_label.place(relx=0.3, rely=0.4, anchor="center")
204. # Input text box
205. self.input\_entry = tk.Text(self.root, width=40, height=5)
206. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
208. # Encrypt button
209. encrypt\_button = tk.Button(self.root, text="Decrypt", command=self.rail\_fence\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
210. encrypt\_button.place(x=400,y=350, anchor="center")
212. # Previous button
213. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
214. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
215. def rail\_fence\_encrypt(self):   #rail fence cipher encryption button and functionality code
216. # Get text from the input entry in the text box
217. rail\_fence\_text = self.input\_entry.get("1.0", "end-1c")
218. self.input\_entry.delete("1.0", "end")
220. key = 3  # Number of rails
221. # Matrix to hold the rail fence cipher
222. rail = [['\n' for i in range(len(rail\_fence\_text))] for j in range(key)]
224. # Direction flag
225. down = False
226. row, col = 0, 0
228. # Fill the rail matrix
229. for char in rail\_fence\_text:
230. # Change direction if at the top or bottom rail
231. if (row == 0) or (row == key - 1):
232. down = not down
234. # Fill the rail matrix
235. rail[row][col] = char
236. col += 1
238. # Move to the next row
239. if down:
240. row += 1
241. else:
242. row -= 1
244. # Construct the cipher text
245. encrypted\_text = ''
246. for i in range(key):
247. for j in range(len(rail\_fence\_text)):
248. if rail[i][j] != '\n':
249. encrypted\_text += rail[i][j]
251. # Open a new window to display output
252. output\_window = tk.Toplevel(self.root)
253. output\_window.title("Encryption Output")
254. output\_window.geometry("400x200")
255. output\_window.config(bg="black")
257. output\_label = tk.Label(output\_window, text="Encrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
258. output\_label.place(relx=0.5, rely=0.3, anchor="center")
260. output\_text = tk.Label(output\_window, text=encrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
261. output\_text.place(relx=0.5, rely=0.5, anchor="center")
263. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(encrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
264. copy\_button.place(relx=0.5, rely=0.8, anchor="center")
266. def keyword\_cipher(self):   #keyword cipher encrypt screen
267. # Clear the root window
268. self.clear\_screen()
270. # New screen for Substitutional Cipher
271. substitution\_label = tk.Label(self.root, text="Keyword Cipher", font=("Times New Roman", 28), fg="white", bg="black")
272. substitution\_label.place(relx=0.5, rely=0.2, anchor="center")
274. # Input box label
275. input\_label = tk.Label(self.root, text="Input Text to be Encrypted:", font=("Times New Roman", 14), fg="white", bg="black")
276. input\_label.place(relx=0.3, rely=0.4, anchor="center")
277. # Input text box
278. self.input\_entry = tk.Text(self.root, width=40, height=5)
279. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
281. # Encrypt button
282. encrypt\_button = tk.Button(self.root, text="Encrypt", command=self.keyword\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
283. encrypt\_button.place(x=400,y=350, anchor="center")
285. # Previous button
286. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_encrypt, font=("Times New Roman", 12), bg="gray", fg="white")
287. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
288. def keyword\_encrypt(self):  #keyword cipher encryption button and functionality code
289. def generate\_keyword\_alphabet(keyword):
290. keyword\_alphabet = ''
291. for char in keyword:
292. if char not in keyword\_alphabet:
293. keyword\_alphabet += char
294. for char in "abcdefghijklmnopqrstuvwxyz":
295. if char not in keyword\_alphabet:
296. keyword\_alphabet += char
297. return keyword\_alphabet
298. def keyword\_cipher\_encrypt(plaintext, keyword):
299. # Generate the keyword alphabet
300. keyword\_alphabet = generate\_keyword\_alphabet(keyword.lower())
301. # Create a dictionary to map each character to its corresponding character in the keyword alphabet
302. mapping = {char: keyword\_alphabet[i] for i, char in enumerate("abcdefghijklmnopqrstuvwxyz")}
303. # Encrypt the plaintext
304. ciphertext = ''.join(mapping[char] if char.isalpha() else char for char in plaintext.lower())
305. return ciphertext
306. # Fixed Keyword
307. keyword = "liveproject"
309. # Get text from the input entry in the text box
310. keyword\_cipher\_text = self.input\_entry.get("1.0", "end-1c")
311. self.input\_entry.delete("1.0", "end")
312. # Encrypt text
313. encrypted\_text =keyword\_cipher\_encrypt(keyword\_cipher\_text, keyword)
315. # Open a new window to display output
316. output\_window = tk.Toplevel(self.root)
317. output\_window.title("Encryption Output")
318. output\_window.geometry("400x200")
319. output\_window.config(bg="black")
321. output\_label = tk.Label(output\_window, text="Encrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
322. output\_label.place(relx=0.5, rely=0.3, anchor="center")
324. output\_text = tk.Label(output\_window, text=encrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
325. output\_text.place(relx=0.5, rely=0.5, anchor="center")
327. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(encrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
328. copy\_button.place(relx=0.5, rely=0.8, anchor="center")

331. def cipher\_types\_decrypt(self): # Decrypt It screen
332. # Clear the root window
333. self.clear\_screen()
335. # New screen for encryption
336. encrypt\_label = tk.Label(self.root, text="Decrypt It", font=("Times New Roman", 30), fg="white", bg="black")
337. encrypt\_label.place(relx=0.5, rely=0.2, anchor="center")
339. # Caesar Cipher button
340. caesar\_button = tk.Button(self.root, text="Caesar Cipher", command=self.caesar\_cipher\_2,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
341. caesar\_button.place(relx=0.35, rely=0.4, anchor="center")
342. # Atbash Cipher button
343. atbash\_button = tk.Button(self.root, text="Atbash Cipher", command=self.atbash\_cipher\_2,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
344. atbash\_button.place(relx=0.65, rely=0.4, anchor="center")
346. # Rail Fence Cipher button
347. rail\_fence\_button = tk.Button(self.root, text="Rail Fence Cipher", command=self.rail\_fence\_cipher\_2,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
348. rail\_fence\_button.place(relx=0.35, rely=0.7, anchor="center")
350. # keyword Cipher button
351. keyword\_button = tk.Button(self.root, text="Keyword Cipher", command=self.keyword\_cipher\_2,width=15, height=3, font=("Times New Roman", 16), bg="gray", fg="white")
352. keyword\_button.place(relx=0.65, rely=0.7, anchor="center")
354. # Previous button
355. previous\_button = tk.Button(self.root, text="Previous", command=self.start, font=("Times New Roman", 12), bg="gray", fg="white")
356. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
358. def caesar\_cipher\_2(self):  #caesar cipher decrypt screen
359. # Clear the root window
360. self.clear\_screen()
362. # New screen for Caesar Cipher
363. caesar\_label = tk.Label(self.root, text="Caesar Cipher", font=("Times New Roman", 28), fg="white", bg="black")
364. caesar\_label.place(relx=0.5, rely=0.2, anchor="center")
366. # Input box label
367. input\_label = tk.Label(self.root, text="Input Text to be Decrypted:", font=("Times New Roman", 14), fg="white", bg="black")
368. input\_label.place(relx=0.3, rely=0.4, anchor="center")
369. # Input text box
370. self.input\_entry = tk.Text(self.root, width=40, height=5)
371. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
373. # Decrypt button
374. encrypt\_button = tk.Button(self.root, text="Decrypt", command=self.caesar\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
375. encrypt\_button.place(x=400,y=350, anchor="center")
377. # Previous button
378. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
379. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
380. def caesar\_decrypt(self):   #caesar cipher decryption button and funcitonality code
381. # Get text from the input entry in the text box
382. caesar\_text = self.input\_entry.get("1.0", "end-1c")
383. self.input\_entry.delete("1.0", "end")
385. shift = 10  # fixing the shift value to 10
386. decrypted\_text = ""
387. for char in caesar\_text:
388. if char.isalpha():  # Check if the character is a letter
389. shifted\_ascii = ord(char) - shift  # Shift the ASCII value backward
390. if char.islower():
391. if shifted\_ascii < ord('a'):
392. shifted\_ascii += 26  # Wrap around if needed
393. elif char.isupper():
394. if shifted\_ascii < ord('A'):
395. shifted\_ascii += 26  # Wrap around if needed
396. decrypted\_text += chr(shifted\_ascii)
397. else:
398. decrypted\_text += char  # Keep non-alphabetic characters unchanged
400. # Open a new window to display output
401. output\_window = tk.Toplevel(self.root)
402. output\_window.title("Encryption Output")
403. output\_window.geometry("400x200")
404. output\_window.config(bg="black")
406. output\_label = tk.Label(output\_window, text="Decrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
407. output\_label.place(relx=0.5, rely=0.3, anchor="center")
409. output\_text = tk.Label(output\_window, text=decrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
410. output\_text.place(relx=0.5, rely=0.5, anchor="center")
412. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(decrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
413. copy\_button.place(relx=0.5, rely=0.8, anchor="center")
415. def atbash\_cipher\_2(self):  #atbash cipher decrypt screen
416. # Clear the root window
417. self.clear\_screen()
419. # New screen for atbash Cipher
420. atbash\_label = tk.Label(self.root, text="atbash Cipher", font=("Times New Roman", 28), fg="white", bg="black")
421. atbash\_label.place(relx=0.5, rely=0.2, anchor="center")
423. # Input box label
424. input\_label = tk.Label(self.root, text="Input Text to be Decrypted:", font=("Times New Roman", 14), fg="white", bg="black")
425. input\_label.place(relx=0.3, rely=0.4, anchor="center")
426. # Input text box
427. self.input\_entry = tk.Text(self.root, width=40, height=5)
428. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
430. # Decrypt button
431. encrypt\_button = tk.Button(self.root, text="Decrypt", command=self.atbash\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
432. encrypt\_button.place(x=400,y=350, anchor="center")
434. # Previous button
435. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
436. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
437. def atbash\_decrypt(self):   #atbash cipher decryption button and funcitonality code
438. # Get text from the input entry in the text box
439. atbash\_text = self.input\_entry.get("1.0", "end-1c")
440. self.input\_entry.delete("1.0", "end")
442. # Define the alphabet
443. alphabet = 'abcdefghijklmnopqrstuvwxyz'
444. reversed\_alphabet = alphabet[::-1]
446. # Convert text to lowercase
447. atbash\_text = atbash\_text.lower()
449. # Initialize an empty string to store the decrypted text
450. decrypted\_text = ''
452. # Decrypt each character in the text
453. for char in atbash\_text:
454. if char.isalpha():
455. # Find the index of the character in the reversed alphabet
456. index = reversed\_alphabet.find(char)
457. # Replace the character with its counterpart in the regular alphabet
458. decrypted\_char = alphabet[index]
459. decrypted\_text += decrypted\_char
460. else:
461. # Keep non-alphabetic characters unchanged
462. decrypted\_text += char
464. # Open a new window to display output
465. output\_window = tk.Toplevel(self.root)
466. output\_window.title("Encryption Output")
467. output\_window.geometry("400x200")
468. output\_window.config(bg="black")
470. output\_label = tk.Label(output\_window, text="Decrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
471. output\_label.place(relx=0.5, rely=0.3, anchor="center")
473. output\_text = tk.Label(output\_window, text=decrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
474. output\_text.place(relx=0.5, rely=0.5, anchor="center")
476. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(decrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
477. copy\_button.place(relx=0.5, rely=0.8, anchor="center")
478. def rail\_fence\_cipher\_2(self):  #rail\_fence cipher decrypt screen
479. # Clear the root window
480. self.clear\_screen()
482. # New screen for rail\_fence Cipher
483. rail\_fence\_label = tk.Label(self.root, text="Rail Fence Cipher", font=("Times New Roman", 28), fg="white", bg="black")
484. rail\_fence\_label.place(relx=0.5, rely=0.2, anchor="center")
486. # Input box label
487. input\_label = tk.Label(self.root, text="Input Text to be Decrypted:", font=("Times New Roman", 14), fg="white", bg="black")
488. input\_label.place(relx=0.3, rely=0.4, anchor="center")
489. # Input text box
490. self.input\_entry = tk.Text(self.root, width=40, height=5)
491. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
493. # Decrypt button
494. encrypt\_button = tk.Button(self.root, text="Decrypt", command=self.rail\_fence\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
495. encrypt\_button.place(x=400,y=350, anchor="center")
497. # Previous button
498. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
499. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
500. def rail\_fence\_decrypt(self):   #rail\_fence cipher decryption button and funcitonality code
501. # Get text from the input entry in the text box
502. rail\_fence\_text = self.input\_entry.get("1.0", "end-1c")
503. self.input\_entry.delete("1.0", "end")
505. key=3
506. # Create a matrix to store the rail fence cipher
507. rail = [['\n' for i in range(len(rail\_fence\_text))] for j in range(key)]
509. # Initialize variables for traversal
510. dir\_down = None
511. row, col = 0, 0
513. # Mark the places with '\*'
514. for i in range(len(rail\_fence\_text)):
515. if row == 0:
516. dir\_down = True
517. if row == key - 1:
518. dir\_down = False
520. # Place the marker
521. rail[row][col] = '\*'
522. col += 1
524. # Move to the next row using the direction flag
525. if dir\_down:
526. row += 1
527. else:
528. row -= 1
530. # Fill the rail matrix
531. index = 0
532. for i in range(key):
533. for j in range(len(rail\_fence\_text)):
534. if (rail[i][j] == '\*') and (index < len(rail\_fence\_text)):
535. rail[i][j] = rail\_fence\_text[index]
536. index += 1
538. # Construct the resultant text by reading the matrix in a zig-zag manner
539. decrypted\_text = []
540. row, col = 0, 0
541. for i in range(len(rail\_fence\_text)):
542. # Check the direction of flow
543. if row == 0:
544. dir\_down = True
545. if row == key - 1:
546. dir\_down = False
548. # Append the character if it's not a marker
549. if rail[row][col] != '\*':
550. decrypted\_text.append(rail[row][col])
551. col += 1
553. # Move to the next row using the direction flag
554. if dir\_down:
555. row += 1
556. else:
557. row -= 1
559. decrypted\_text= "".join(decrypted\_text)
561. # Open a new window to display output
562. output\_window = tk.Toplevel(self.root)
563. output\_window.title("Encryption Output")
564. output\_window.geometry("400x200")
565. output\_window.config(bg="black")
567. output\_label = tk.Label(output\_window, text="Decrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
568. output\_label.place(relx=0.5, rely=0.3, anchor="center")
570. output\_text = tk.Label(output\_window, text=decrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
571. output\_text.place(relx=0.5, rely=0.5, anchor="center")
573. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(decrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
574. copy\_button.place(relx=0.5, rely=0.8, anchor="center")
575. def keyword\_cipher\_2(self):  #keyword cipher decrypt screen
576. # Clear the root window
577. self.clear\_screen()
579. # New screen for keyword Cipher
580. keyword\_label = tk.Label(self.root, text="Keyword Cipher", font=("Times New Roman", 28), fg="white", bg="black")
581. keyword\_label.place(relx=0.5, rely=0.2, anchor="center")
583. # Input box label
584. input\_label = tk.Label(self.root, text="Input Text to be Decrypted:", font=("Times New Roman", 14), fg="white", bg="black")
585. input\_label.place(relx=0.3, rely=0.4, anchor="center")
586. # Input text box
587. self.input\_entry = tk.Text(self.root, width=40, height=5)
588. self.input\_entry.place(relx=0.5, rely=0.5, anchor="center")
590. # Decrypt button
591. encrypt\_button = tk.Button(self.root, text="Decrypt", command=self.keyword\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
592. encrypt\_button.place(x=400,y=350, anchor="center")
594. # Previous button
595. previous\_button = tk.Button(self.root, text="Previous", command=self.cipher\_types\_decrypt, font=("Times New Roman", 12), bg="gray", fg="white")
596. previous\_button.place(relx=0.1, rely=0.1, anchor="center")
597. def keyword\_decrypt(self):   #keyword cipher decryption button and funcitonality code
598. def generate\_keyword\_alphabet(keyword):
599. keyword\_alphabet = ''
600. for char in keyword:
601. if char not in keyword\_alphabet:
602. keyword\_alphabet += char
603. for char in "abcdefghijklmnopqrstuvwxyz":
604. if char not in keyword\_alphabet:
605. keyword\_alphabet += char
606. return keyword\_alphabet
607. def keyword\_cipher\_decrypt(ciphertext, keyword):
608. # Generate the keyword alphabet
609. keyword\_alphabet = generate\_keyword\_alphabet(keyword.lower())
611. # Create a dictionary to map each character in the keyword alphabet to its corresponding
612. # character in the standard alphabet
613. mapping = {keyword\_alphabet[i]: char for i, char in enumerate("abcdefghijklmnopqrstuvwxyz")}
615. # Decrypt the ciphertext
616. plaintext = ''.join(mapping[char] if char.isalpha() else char for char in ciphertext.lower())
618. return plaintext
620. # Fixed Keyword
621. keyword = "liveproject"
623. # Get text from the input entry in the text box
624. keyword\_cipher\_text = self.input\_entry.get("1.0", "end-1c")
625. self.input\_entry.delete("1.0", "end")
626. # Encrypt text
627. decrypted\_text =keyword\_cipher\_decrypt(keyword\_cipher\_text, keyword)
629. # Open a new window to display output
630. output\_window = tk.Toplevel(self.root)
631. output\_window.title("Encryption Output")
632. output\_window.geometry("400x200")
633. output\_window.config(bg="black")
635. output\_label = tk.Label(output\_window, text="Encrypted Text", font=("Times New Roman", 20), fg="white", bg="black")
636. output\_label.place(relx=0.5, rely=0.3, anchor="center")
638. output\_text = tk.Label(output\_window, text=decrypted\_text, font=("Times New Roman", 14), fg="white", bg="black")
639. output\_text.place(relx=0.5, rely=0.5, anchor="center")
641. copy\_button = tk.Button(output\_window, text="Copy", command=lambda: self.copy\_to\_clipboard(decrypted\_text), font=("Times New Roman", 12), bg="gray", fg="white")
642. copy\_button.place(relx=0.5, rely=0.8, anchor="center")

645. def copy\_to\_clipboard(self, text):  #function to copy resultant text
646. pyperclip.copy(text)
648. def clear\_screen(self): #function to clear the screen
649. # Clear the root window
650. for widget in self.root.winfo\_children():
651. widget.destroy()
652. root = tk.Tk()
653. app = CipherCraftApp(root)
654. root.mainloop()
655. **Conclusion**

**7.1 Summary**

In conclusion, CipherCraft represents a comprehensive exploration of cryptography principles and their practical application in a user-friendly software tool. Through the implementation of classical and modern ciphers, including the Caesar Cipher, Atbash Cipher, Rail Fence Cipher, and Keyword Cipher, users are introduced to various encryption techniques and their underlying mechanisms. The project also sheds light on the importance of key management and the role of cryptography in securing digital communications.

Additionally, CipherCraft leverages Python and the Tkinter library to provide an intuitive user interface for encrypting and decrypting messages. The use of Pyperclip facilitates seamless clipboard integration, enhancing user experience and convenience. Development tools such as Visual Studio Code and ChatGPT have been instrumental in the project's creation, offering robust code editing capabilities and AI-powered assistance.

Through this project, users not only gain practical experience in cryptography but also develop critical thinking skills by exploring the strengths and weaknesses of different encryption methods. Moving forward, CipherCraft serves as a valuable educational resource and a testament to the power of cryptography in safeguarding sensitive information in the digital age.

**7.2 Lesson Learned**

Through the development of CipherCraft, several valuable lessons have been learned. Firstly, the project highlighted the importance of understanding cryptography fundamentals, including various encryption techniques and their real-world applications. This involved delving into classical ciphers like the Caesar Cipher and modern algorithms like the Advanced Encryption Standard (AES).

Secondly, effective project management practices were essential for coordinating tasks, setting deadlines, and ensuring progress. Clear communication among team members facilitated collaboration and problem-solving, ultimately contributing to the project's success.

Moreover, the significance of user feedback became evident during the testing phase. Iterative improvements based on user input helped enhance the software's usability and functionality, emphasizing the importance of user-centric design principles.

Furthermore, the project underscored the value of leveraging existing libraries and tools to streamline development processes. Utilizing Python libraries such as Tkinter for the user interface and Pyperclip for clipboard integration saved time and effort, demonstrating the efficiency of leveraging available resources.

Overall, CipherCraft served as a valuable learning experience, reinforcing the importance of cryptography in cybersecurity, project management best practices, user-centered design, and the effective utilization of development tools and libraries. These lessons will undoubtedly inform future projects and contribute to ongoing professional growth.

1. **References**

* <https://www.geeksforgeeks.org/cryptography-and-its-types/>
* <https://www.geeksforgeeks.org/cryptography-tutorial/>
* <https://www.geeksforgeeks.org/cryptography-introduction/>
* <https://docs.oracle.com/cd/E19047-01/sunscreen151/806-5397/i996724/index.html#:~:text=Encryption%20is%20the%20process%20by,is%20called%20the%20plaintext%20message>.
* <https://www.geeksforgeeks.org/basics-of-cryptographic-algorithms/>
* <https://en.wikipedia.org/wiki/Caesar_cipher>
* <https://www.khanacademy.org/computing/computer-science/cryptography/crypt/v/caesar-cipher#:~:text=The%20Caesar%20Cipher%2C%20used%20by,exploits%20patterns%20in%20letter%20frequencies>.
* <https://www.geeksforgeeks.org/implementing-atbash-cipher/>
* <https://en.wikipedia.org/wiki/Atbash>
* <https://www.dcode.fr/atbash-cipher>
* <https://crypto.interactive-maths.com/rail-fence-cipher.html>
* <https://www.geeksforgeeks.org/rail-fence-cipher-encryption-decryption/>
* <https://www.geeksforgeeks.org/keyword-cipher/>
* <https://braingle.com/brainteasers/codes/keyword.php>