

National University

of ScienceS & Technology

CS484-Information security

bese-12 2k21 ab

Assignment 1

Submitted To: Prof. Hirra Anwar

Submitted By:

|  |  |
| --- | --- |
| Name | CMS |
| Rana Mahad Ahmer | 367679 |

Submission Date: 12-10-2024

Contents

[**Assignment:** 2](#_Toc179652381)

[**-Task 1-** 2](#_Toc179652382)

[**Part (a): (Code in Codr.rar/Task1/PartA\_Encryption.py)** 2](#_Toc179652383)

[**Part (b): (Code in Codr.rar/Task1/PartB\_Decryption.py)** 3](#_Toc179652384)

[**Part (c): (Code in Codr.rar/Task2/PartC\_EN\_DE\_Case\_Insensitive.py)** 5](#_Toc179652385)

[**-Task 2-** 7](#_Toc179652386)

[**Part (a): (Code in Codr.rar/Task2/PartA\_Encryption.py)** 7](#_Toc179652387)

[**Part (b): (Code in Codr.rar/Task2/PartB\_Decryption.py)** 8](#_Toc179652388)

[**Part (c): (Code in Codr.rar/Task2/PartC\_EN\_DE\_Valid\_Chars.py)** 9](#_Toc179652389)

[**-GitHub Link-** 11](#_Toc179652390)

# **Assignment:**

## **-Task 1-**

### **Part (a): (Code in Codr.rar/Task1/PartA\_Encryption.py)**

Write code to **encrypt** the message “Even the strongest encryption can be undone by the faintest error” using the following 5x5 Polybius square as the encryption key.

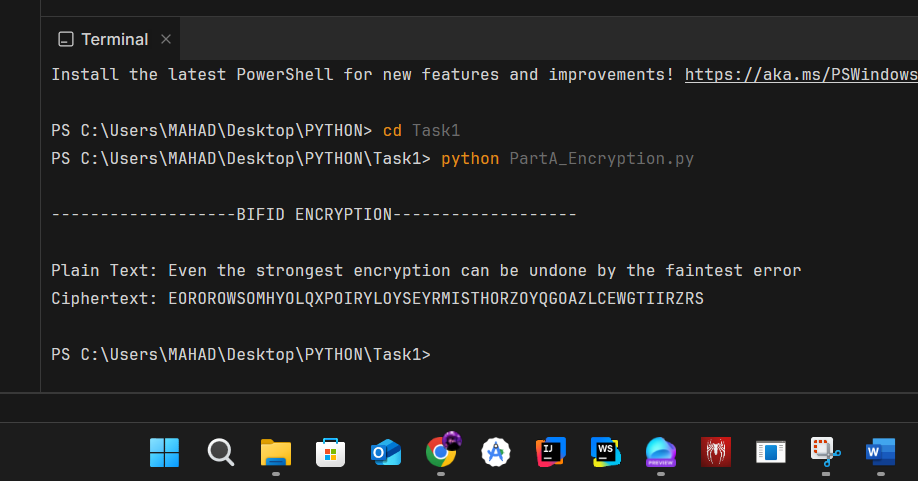
#### **Assumptions/Note:**

Since the Polybius Square was given for capital case only so produced cipher text is also capital case and given Polybius Square does not contain space, so spaces will not be included in the cipher so when cipher is decrypted the text will be capital case and without spaces.

#### **Code:**

|  |
| --- |
| polybius\_square = {  'B': (1, 1), 'G': (1, 2), 'W': (1, 3), 'K': (1, 4), 'Z': (1, 5),  'Q': (2, 1), 'P': (2, 2), 'N': (2, 3), 'D': (2, 4), 'S': (2, 5),  'I': (3, 1), 'O': (3, 2), 'A': (3, 3), 'X': (3, 4), 'E': (3, 5),  'F': (4, 1), 'C': (4, 2), 'L': (4, 3), 'U': (4, 4), 'M': (4, 5),  'T': (5, 1), 'H': (5, 2), 'Y': (5, 3), 'V': (5, 4), 'R': (5, 5)  }  reverse\_square = {v: k for k, v in polybius\_square.items()}  def encrypt(plain\_text):  plain\_text = plain\_text.replace("J", "I").upper().replace(" ", "")  coordinates = []  for char in plain\_text:  if char in polybius\_square:  coordinates.append(polybius\_square[char])  rows = [row for row, col in coordinates]  cols = [col for row, col in coordinates]  merged = rows + cols  produced\_cipher\_text = ""  for i in range(0, len(merged), 2):  pair = (merged[i], merged[i + 1])  produced\_cipher\_text += reverse\_square[pair]  return produced\_cipher\_text  print("\n-------------------BIFID ENCRYPTION-------------------\n")  plain\_text = "Even the strongest encryption can be undone by the faintest error"  cipher\_text = encrypt(plain\_text)  print(f"Plain Text: {plain\_text}")  print(f"Ciphertext: {cipher\_text}\n") |

#### **Output:**



#### **Text:**

##### **Plain Text:**

Even the strongest encryption can be undone by the faintest error

**Ciphertext:** EOROROWSOMHYOLQXPOIRYLOYSEYRMISTHORZOYQGOAZLCEWGTIIRZRS

### **Part (b): (Code in Codr.rar/Task1/PartB\_Decryption.py)**

Write code to **decrypt** the ciphertext generated in part (a).

#### **Assumptions:**

Since the Polybius Square was given for capital case only so produced cipher text is also capital case and given Polybius Square does not contain space, so spaces will not be included in the cipher so when cipher is decrypted the text will be capital case and without spaces.

#### **Code:**

|  |
| --- |
| polybius\_square = {  'B': (1, 1), 'G': (1, 2), 'W': (1, 3), 'K': (1, 4), 'Z': (1, 5),  'Q': (2, 1), 'P': (2, 2), 'N': (2, 3), 'D': (2, 4), 'S': (2, 5),  'I': (3, 1), 'O': (3, 2), 'A': (3, 3), 'X': (3, 4), 'E': (3, 5),  'F': (4, 1), 'C': (4, 2), 'L': (4, 3), 'U': (4, 4), 'M': (4, 5),  'T': (5, 1), 'H': (5, 2), 'Y': (5, 3), 'V': (5, 4), 'R': (5, 5)  }  def decrypt(ciphertext):  coords = []  for char in ciphertext:  if char in polybius\_square:  row, col = polybius\_square[char]  coords.append(row)  coords.append(col)  center = len(coords) // 2  row\_coords = coords[:center]  col\_coords = coords[center:]  plain\_text = ""  for i in range(center):  row = row\_coords[i]  col = col\_coords[i]  for key, value in polybius\_square.items():  if value == (row, col):  plain\_text += key  break  return plain\_text  print("\n-------------------BIFID DECRYTPTION-------------------\n")  cipher\_text = "EOROROWSOMHYOLQXPOIRYLOYSEYRMISTHORZOYQGOAZLCEWGTIIRZRS"  decrypted\_text = decrypt(cipher\_text)  print(f"CipherText: {cipher\_text}")  print(f"DecrypText: {decrypted\_text}\n") |

#### **Output:**

A screenshot of a computer program

Description automatically generated

#### **Text:**

**CipherText:** EOROROWSOMHYOLQXPOIRYLOYSEYRMISTHORZOYQGOAZLCEWGTIIRZRS

**DecrypText:** EVENTHESTRONGESTENCRYPTIONCANBEUNDONEBYTHEFAINTESTERROR

### **Part (c): (Code in Codr.rar/Task2/PartC\_EN\_DE\_Case\_Insensitive.py)**

Modify your code to handle the inclusion of both **uppercase and lowercase** letters while maintaining consistency in encryption and decryption. The code should treat uppercase (A-Z) and lowercase (a-z) letters as equivalent when performing encryption and decryption. For example, both "A" and "a" should be encrypted to the same letter in the ciphertext.

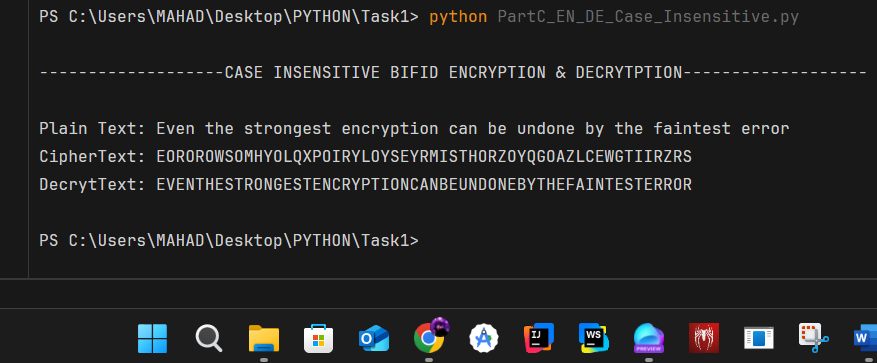
#### **Assumptions:**

Since the Polybius Square was given for capital case only and for handling I have made all the letters given in plain text capital so that Polybius Square matches perfectly and produced cipher text is also capital case and given that Polybius Square does not contain space, spaces will not be included in the cipher so when cipher is decrypted the text will be capital case and without space.

#### **Code:**

|  |
| --- |
| polybius\_square = {  'B': (1, 1), 'G': (1, 2), 'W': (1, 3), 'K': (1, 4), 'Z': (1, 5),  'Q': (2, 1), 'P': (2, 2), 'N': (2, 3), 'D': (2, 4), 'S': (2, 5),  'I': (3, 1), 'O': (3, 2), 'A': (3, 3), 'X': (3, 4), 'E': (3, 5),  'F': (4, 1), 'C': (4, 2), 'L': (4, 3), 'U': (4, 4), 'M': (4, 5),  'T': (5, 1), 'H': (5, 2), 'Y': (5, 3), 'V': (5, 4), 'R': (5, 5)  }  reverse\_square = {v: k for k, v in polybius\_square.items()}  def encrypt(plain\_text):  plain\_text = plain\_text.replace("J", "I").upper().replace(" ", "")  coordinates = []  for char in plain\_text:  if char in polybius\_square:  coordinates.append(polybius\_square[char])  rows = [row for row, col in coordinates]  cols = [col for row, col in coordinates]  merged = rows + cols  produced\_cipher\_text = ""  for i in range(0, len(merged), 2):  pair = (merged[i], merged[i + 1])  produced\_cipher\_text += reverse\_square[pair]  return produced\_cipher\_text  def decrypt(ciphertext):  coords = []  for char in ciphertext:  if char in polybius\_square:  row, col = polybius\_square[char]  coords.append(row)  coords.append(col)  center = len(coords) // 2  row\_coords = coords[:center]  col\_coords = coords[center:]  plain\_text = ""  for i in range(center):  row = row\_coords[i]  col = col\_coords[i]  for key, value in polybius\_square.items():  if value == (row, col):  plain\_text += key  break  return plain\_text  print("\n-------------------CASE INSENSITIVE BIFID ENCRYPTION & DECRYTPTION----------------------\n")  plain\_text = "Even the strongest encryption can be undone by the faintest error"  cipher\_text = encrypt(plain\_text.upper())  print(f"Plain Text: {plain\_text}")  print(f"CipherText: {cipher\_text}")  decrypted\_text = decrypt(cipher\_text)  print(f"DecrytText: {decrypted\_text}\n") |

#### **Output:**



#### **Text:**

##### **Plain Text:**

Even the strongest encryption can be undone by the faintest error

**CipherText:** EOROROWSOMHYOLQXPOIRYLOYSEYRMISTHORZOYQGOAZLCEWGTIIRZRS

**DecrytText:** EVENTHESTRONGESTENCRYPTIONCANBEUNDONEBYTHEFAINTESTERROR

## **-Task 2-**

### **Part (a): (Code in Codr.rar/Task2/PartA\_Encryption.py)**

Write a Python program to **encrypt** the message: "Complexity in cryptography adds layers of intrigue and security" using the keyword "SECURITY".

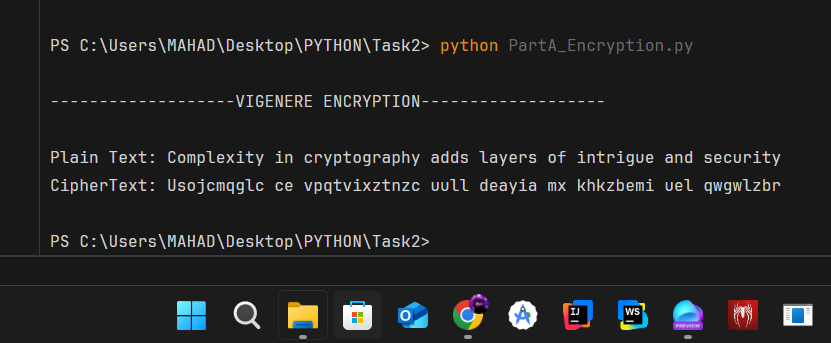
#### **Assumptions:**

The plain text and keyword contain only alphabetical characters.

#### **Code:**

|  |
| --- |
| def encrypt(plain\_text, key):  cipher\_text = ""  keyword\_repeated = (key \* (len(plain\_text) // len(key) + 1))[:len(plain\_text)]  for i in range(len(plain\_text)):  if plain\_text[i] == " ":  cipher\_text += plain\_text[i]  else:  char\_base = ord('a') if plain\_text[i].islower() else ord('A')  key\_base = ord('a') if keyword\_repeated[i].islower() else ord('A')  letter\_shift = (ord(plain\_text[i]) - char\_base + ord(keyword\_repeated[i]) - key\_base) % 26  cipher\_text += str(chr(letter\_shift + char\_base))  return cipher\_text  plain\_text = "Complexity in cryptography adds layers of intrigue and security"  key = "SECURITY"  cipher\_text = encrypt(plain\_text, key)  print("\n-------------------VIGENERE ENCRYPTION-------------------\n")  print(f"Plain Text: {plain\_text}")  print(f"CipherText: {cipher\_text}\n") |

#### **Output:**



#### **Text:**

##### **Plain Text:**

Complexity in cryptography adds layers of intrigue and security

##### **CipherText:**

Usojcmqglc ce vpqtvixztnzc uull deayia mx khkzbemi uel qwgwlzbr

### **Part (b): (Code in Codr.rar/Task2/PartB\_Decryption.py)**

Write a Python program to **decrypt** the ciphertext generated in part (a) using the same keyword

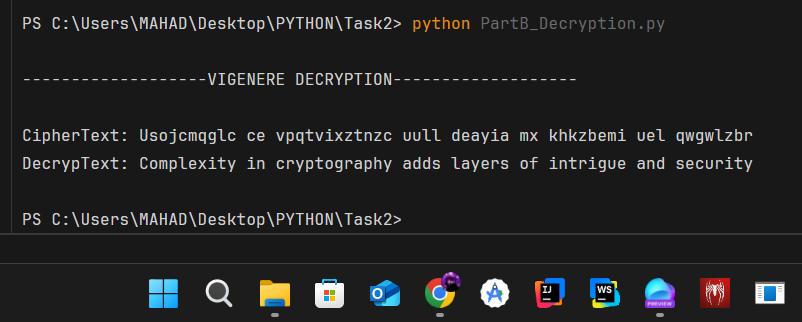
#### **Assumptions:**

The cipher text and keyword contain only alphabetical characters.

#### **Code:**

|  |
| --- |
| def decrypt(cipher\_text, key):  decrypted\_text = ""  keyword\_repeated = (key \* (len(cipher\_text) // len(key) + 1))[:len(cipher\_text)]  for i in range(len(cipher\_text)):  if cipher\_text[i] == " ":  decrypted\_text += cipher\_text[i]  else:  char\_base = ord('a') if cipher\_text[i].islower() else ord('A')  key\_base = ord('a') if keyword\_repeated[i].islower() else ord('A')  letter\_shift = (ord(cipher\_text[i]) - char\_base - (ord(keyword\_repeated[i]) - key\_base)) % 26  decrypted\_text += str(chr(letter\_shift + char\_base))  return decrypted\_text  cipher\_text = "Usojcmqglc ce vpqtvixztnzc uull deayia mx khkzbemi uel qwgwlzbr"  key = "SECURITY"  decrypted\_text = decrypt(cipher\_text, key)  print("\n-------------------VIGENERE DECRYPTION-------------------\n")  print(f"CipherText: {cipher\_text}")  print(f"DecrypText: {decrypted\_text}\n") |

#### **Output:**



#### **Text:**

##### **CipherText:**

Usojcmqglc ce vpqtvixztnzc uull deayia mx khkzbemi uel qwgwlzbr

##### **DecrypText:**

Complexity in cryptography adds layers of intrigue and security

### **Part (c): (Code in Codr.rar/Task2/PartC\_EN\_DE\_Valid\_Chars.py)**

Modify your program to include a feature that allows the user to input their own message and keyword for encryption and decryption. Ensure the program validates the input (e.g., checks for invalid characters)

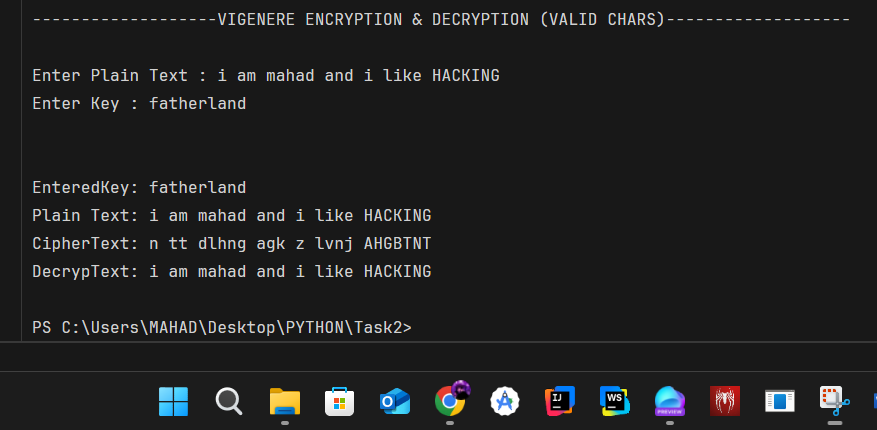
#### **Assumptions:**

To force user, I have used a while loop that will keep asking user for valid Plain Text and Key forever, if user enters some characters other than alphabets the loop will continue asking user for valid input. Furthermore, users can enter space in plain text but not in the key.

#### **Code:**

|  |
| --- |
| def encrypt(plain\_text, key):  cipher\_text = ""  keyword\_repeated = (key \* (len(plain\_text) // len(key) + 1))[:len(plain\_text)]  for i in range(len(plain\_text)):  if plain\_text[i] == " ":  cipher\_text += plain\_text[i]  else:  char\_base = ord('a') if plain\_text[i].islower() else ord('A')  key\_base = ord('a') if keyword\_repeated[i].islower() else ord('A')  letter\_shift = (ord(plain\_text[i]) - char\_base + ord(keyword\_repeated[i]) - key\_base) % 26  cipher\_text += str(chr(letter\_shift + char\_base))  return cipher\_text  def decrypt(cipher\_text, key):  decrypted\_text = ""  keyword\_repeated = (key \* (len(cipher\_text) // len(key) + 1))[:len(cipher\_text)]  for i in range(len(cipher\_text)):  if cipher\_text[i] == " ":  decrypted\_text += cipher\_text[i]  else:  char\_base = ord('a') if cipher\_text[i].islower() else ord('A')  key\_base = ord('a') if keyword\_repeated[i].islower() else ord('A')  letter\_shift = (ord(cipher\_text[i]) - char\_base - (ord(keyword\_repeated[i]) - key\_base)) % 26  decrypted\_text += str(chr(letter\_shift + char\_base))  return decrypted\_text  plain\_text = ""  # ""Complexity in cryptography adds layers of intrigue and security"  key = ""  # ""SECURITY"  print("\n-------------------VIGENERE ENCRYPTION & DECRYPTION (VALID CHARS)-------------------\n")  while (True):  plain\_text = input("Enter Plain Text : ")  if plain\_text.replace(" ", "").isalpha():  break  else:  print("Enter only alphabets\n")  continue  while (True):  key = input("Enter Key : ")  if key.isalpha():  break  else:  print("Enter only alphabets\n")  continue  print(f"\n\nEnteredKey: {key}")  print(f"Plain Text: {plain\_text}")  cipher\_text = encrypt(plain\_text, key)  print(f"CipherText: {cipher\_text}")  decrypted\_text = decrypt(cipher\_text, key)  print(f"DecrypText: {decrypted\_text}\n") |

#### **Output:**



#### **Text:**

Enter Plain Text : i am mahad and i like HACKING

Enter Key : fatherland

**EnteredKey**:

fatherland

##### **Plain Text:**

i am mahad and i like HACKING

##### **CipherText:**

n tt dlhng agk z lvnj AHGBTNT

##### **DecrypText:**

i am mahad and i like HACKING

## **-GitHub Link-**

Zip File is also attached:

[https://github.com/RanaMahadAhmer/IS\_Assignment\_1](https://github.com/RanaMahadAhmer/IS_Assignment_1 )