CSS EXP1 = Implementation of a Product Cipher Using Substitution & Transposition Ciphers

YASH ASHOK SHIRSATH TE AI&DS - 74 / 201101006

```
# Implementation of a Product Cipher Using Substitution Ciphers - Additive Cipher Encryption
def encrypt_additive(plaintext, key):
  """Encrypts plaintext using an additive cipher with the given key."""
  ciphertext = ""
  for char in plaintext:
    if char.isalpha():
      base = ord('A') if char.isupper() else ord('a')
      new_char_code = (ord(char) - base + key) % 26 + base
      new_char = chr(new_char_code)
      ciphertext += new_char
    else:
      ciphertext += char
  return ciphertext

plaintext = "ENEMY ATTACK TODAY" # PT & KEY
key = 15
ciphertext = encrypt_additive(plaintext, key) # Encrypt the PT
print("PT TO CIPHERTEXT:-", ciphertext)


# Implementation of a Product Cipher Using Substitution Ciphers - Additive Cipher Decryption
def decrypt_additive(ciphertext, key):
  """Decrypts ciphertext using an additive cipher with the given key."""
  plaintext = ""
  for char in ciphertext:
    if char.isalpha():
      new_char_code = (ord(char) - ord('A') - key + 26) % 26 + ord('A')
      plaintext += chr(new_char_code)
    else:
      plaintext += char
  return plaintext

plaintext = decrypt_additive(ciphertext, key)  # Decrypt the CT
print("CT TO PLAINTEXT:-", plaintext)
```

```
PT TO CIPHERTEXT:- TCTBN PIIPRZ IDSPN
CT TO PLAINTEXT:- ENEMY ATTACK TODAY
```

```
# Implementation of a Product Cipher Using Substitution Ciphers - Multiplicative Cipher Encryption & Decryptic
def encrypt_multiplicative(plaintext, key):
  """Encrypts plaintext using a multiplicative cipher with the given key."""
  ciphertext = ""
  for char in plaintext:
    if char.isalpha():
      base = ord('A') if char.isupper() else ord('a')
      new_char_code = (ord(char) - base) * key % 26 + base
      new_char = chr(new_char_code)
      ciphertext += new_char
    else:
      ciphertext += char
  return ciphertext

def decrypt_multiplicative(ciphertext, key):
  """Decrypts ciphertext using a multiplicative cipher with the given key."""
  plaintext = ""
  for char in ciphertext:
    if char.isalpha():
      base = ord('A') if char.isupper() else ord('a')
      multiplicative_inverse = find_multiplicative_inverse(key, 26)
      new_char_code = (ord(char) - base) * multiplicative_inverse % 26 + base
      new_char = chr(new_char_code)
      plaintext += new_char
    else:
      plaintext += char
  return plaintext

def find_multiplicative_inverse(a, m):
  """Finds the modular multiplicative inverse of 'a' modulo 'm' using the extended Euclidean algorithm."""
  m0 = m
  y = 0
  x = 1
```

```
  if m == 1:
      return 0
  while a > 1:
      # q is quotient
      q = a // m
      t = m
      m = a % m
      a = t
      t = y
      y = x - q * y
      x = t
  if x < 0:
      x = x + m0
  return x

plaintext = "ENEMY ATTACK TODAY"
key = 5

ciphertext = encrypt_multiplicative(plaintext, key)
print("PT TO CIPHERTEXT:-", ciphertext)

decrypted_text = decrypt_multiplicative(ciphertext, key)
```

```
PT TO CIPHERTEXT:- UNUIQ ARRAKY RSPAQ
CT TO PLAINTEXT:- ENEMY ATTACK TODAY
```

```
# Implementation of a Product Cipher Using Transposition Ciphers - Keyless
def encrypt_rail_fence(plaintext, num_rails):
  """Encrypts plaintext using a Rail Fence cipher with the given number of rails."""
  ciphertext = ""
  rail = 0
  direction_down = True

  matrix = [[None for _ in range(len(plaintext))] for _ in range(num_rails)]
  for i in range(len(plaintext)):
    matrix[rail][i] = plaintext[i]
    if rail == 0:
      direction_down = True
    elif rail == num_rails - 1:
      direction_down = False
    rail += 1 if direction_down else -1

  for row in matrix:
    ciphertext += "".join(char for char in row if char is not None)

  return ciphertext

def decrypt_rail_fence(ciphertext, num_rails):
  """Decrypts ciphertext using a Rail Fence cipher with the given number of rails.""
  plaintext = ""
  rail = 0
  direction_down = True

  matrix = [[None for _ in range(len(ciphertext))] for _ in range(num_rails)]
  for i in range(len(ciphertext)):
    matrix[rail][i] = "*"
    if rail == 0:
      direction_down = True
    elif rail == num_rails - 1:
      direction_down = False
    rail += 1 if direction_down else -1

  cipher_index = 0
  for row in matrix:
    for i in range(len(row)):
      if row[i] == "*" and cipher_index < len(ciphertext):
        row[i] = ciphertext[cipher_index]
        cipher_index += 1

  for i in range(len(ciphertext)):
    for row in matrix:
      if row[i] is not None:
        plaintext += row[i]
        break

  return plaintext

plaintext = "ENEMY ATTACK TODAY"
num_rails = 3

ciphertext = encrypt_rail_fence(plaintext, num_rails)
print("PT TO CIPHERTEXT:-", ciphertext)
```

```
decrypted_text = decrypt_rail_fence(ciphertext, num_rails)
```

```
PT TO CIPHERTEXT:- EYT ANM TAKTDYEACO
CT TO PLAINTEXT:- ENEMY ATTACK TODAY
```

```
# Implementation of a Product Cipher Using Transposition Ciphers - With Key
import math
key = "HACK"

def encryptMessage(msg):
    cipher = ""

    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    col = len(key)

    row = int(math.ceil(msg_len / col))
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)
    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]

    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                          for row in matrix])
        k_indx += 1

    return cipher

def decryptMessage(cipher):
    msg = ""

    k_indx = 0
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)
    col = len(key)
    row = int(math.ceil(msg_len / col))
    key_lst = sorted(list(key))
    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])

        for j in range(row):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
        k_indx += 1

    try:
        msg = ''.join(sum(dec_cipher, []))
    except TypeError:
        raise TypeError("This program cannot",
                        "handle repeating words.")

    null_count = msg.count('_')

    if null_count > 0:
        return msg[: -null_count]

    return msg

msg = "ENEMY ATTACK TODAY"

cipher = encryptMessage(msg)
print("PT TO CIPHERTEXT:- {}".
               format(cipher))

print("CT TO PLAINTEXT:- {}".
       format(decryptMessage(cipher)))
```

```
PT TO CIPHERTEXT:- N ATYEACO_EYT AMTKD_
CT TO PLAINTEXT:- ENEMY ATTACK TODAY
```