

EXPERIMENT NO-2**IMPLEMENTATION OF PRODUCT CIPHER USING SUBSTITUTION AND TRANSPOSITION TECHNIQUE****Date of Performance: 30/07/2021****Date of Submission: 13/08/2021****1 a] Vigenere Cipher:****Aim:**

To perform vignere cipher encryption, product cipher using vignere encryption and railfence cipher and subsequent product cipher using railfence encryption.

Theory:

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets .The encryption of the original text is done using the Vigenère square or Vigenère table.

- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

Algorithm:

Encryption algorithm:

1. Accept plain text and key as input.
2. Initialize an empty list for cipher text
3. Repeat steps for all the characters
4. Add ord of plain text character with ord of key and modulo 26
5. Add ord of "A" to it
6. Append it to cipher text

Decryption Algorithm:

1. Accept cipher text and key as input.
2. Initialize an empty list for plain text
3. Repeat steps for all the characters
4. Subtract ord of cipher text character with ord of key and modulo 26
5. Add ord of "A" to it
6. Append it to plain text

Code:

```
# Vigenere Cipher
```

```
## This function generates the key in a cyclic manner until it's length isn't #equal to the length of original text
```

```
def generateKey(string, key):
```

```
    key = list(key)
```

```
    if len(string) == len(key):
```

```
        return(key)
```

```
    else:
```

```
        for i in range(len(string) -  
                        len(key)):
```

```
            key.append(key[i % len(key)])
```

```
    return("".join(key))
```

```
# This function returns the encrypted text generated with the help of the #key
```

```
def cipherText(string, key):
```

```
    cipher_text = []
```

```
    for i in range(len(string)):
```

```
        x = (ord(string[i]) +  
             ord(key[i])) % 26
```

```
        x += ord('A')
```

```
        cipher_text.append(chr(x))
```

```
    return("".join(cipher_text))
```

```
# This function decrypts the encrypted text and returns the original text
```

```
def originalText(cipher_text, key):
```

```
    orig_text = []
```

```
    for i in range(len(cipher_text)):
```

```
x = (ord(cipher_text[i]) -  
      ord(key[i]) + 26) % 26  
x += ord('A')  
orig_text.append(chr(x))  
return("".join(orig_text))
```

Driver code

```
if __name__ == "__main__":  
    string = "XIEISBEST"  
    keyword = "MUMBAI"  
    key = generateKey(string, keyword)  
    cipher_text = cipherText(string, key)  
    print("Ciphertext :", cipher_text)  
    print("Original/Decrypted Text :",  
          originalText(cipher_text, key))
```

Output:

```
Ciphertext : JCQJSJQMF  
Original/Decrypted Text : XIEISBEST
```

1b] Product Cipher Encryption and Decryption using Vigenere Theory:

In cryptography, a product cipher combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components to make it resistant to cryptanalysis. The product cipher combines a sequence of simple transformations such as substitution (S-box), permutation (P-box), and modular arithmetic.

Encryption

Virgere Cipher:

The plaintext(P) and key(K) are added modulo 26.

$$E_i = (P_i + K_i) \bmod 26$$

Product Cipher:

$$EPR_i = (E_i + K_i) \bmod 26$$

Decryption

$$DPR_i = (EPR_i - K_i + 26) \bmod 26$$

$$D_i = (DPR_i - K_i + 26) \bmod 26$$

Algorithm:**Encryption algorithm:**

1. Accept plain text and key as input.
2. Initialize an empty list for cipher text
3. Repeat steps for all the characters
4. Add ord of plain text character with ord of key and modulo 26
5. Add ord of "A" to it
6. Append it to cipher text
7. Repeat steps 1 to 6 again with cipher text and key as input.

Decryption Algorithm:

1. Accept product cipher text and key as input.
2. Initialize an empty list for plain text
3. Repeat steps for all the characters
4. Subtract ord of product cipher text character with ord of key and modulo 26
5. Add ord of "A" to it
6. Append it to plain text
7. Repeat steps 1 to 6 again for previously obtained plain text and key

Code:

```
# Vigenere Cipher
## This function generates the key in a cyclic manner until it's length isn't
#equal to the length of original
text def generateKey(string,
key):
    key = list(key)
```

```
if
len(string) == len(key):

return(key)    else:
    for i in range(len(string) -
                    len(key)):
        key.append(key[i % len(key)])
    return("".join(key))

# This function returns the encrypted text generated with the help of the #key def
cipherText(string, key):
    cipher_text = []
    for i in range(len(string)):
        x = (ord(string[i]) +
             ord(key[i])) % 26          x
        += ord('A')
    cipher_text.append(chr(x))
    return("".join(cipher_text))

# This function decrypts the encrypted text and returns the original text
def originalText(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) -
             ord(key[i]) + 26) % 26
        x += ord('A')
    orig_text.append(chr(x))
    return("".join(orig_text))

# Driver code
if __name__
"__main__":
    string =
    "XIEISBEST"
    keyword =
    "MUMBAI"
    key = generateKey(string, keyword)
    cipher_text = cipherText(string,key)
    productCipher = cipherText(cipher_text,key)
    productDecrypt = originalText(productCipher,key)
    original_text = originalText(cipher_text, key)
    print("Vigenere Cipher \nCiphertext :", cipher_text)
    print("Product Cipher\nCipher text: ",productCipher)
    print("Product cipher \nDecrypted text",productDecrypt)
    print("Original/Decrypted Text :", original_text)
```

Output:

```
Vigenere Cipher
Ciphertext : JCQJSJQMF
Product Cipher
Cipher text: VWCKSRCGR
Product cipher
Decrypted text JCQJSJQMF
Original/Decrypted Text : XIEISBEST
```

TRANSPOSITION TECHNIQUE:**2a] Railfence Technique****Theory:**

In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

```
A U T H O R
1 6 5 2 3 4
-----
W E A R E D
I S C O V E
R E D S A V
E Y O U R S
E L F A B C
```

yields the cipher

WIREEROSUA EVARBDEVSCACDOFESEYL.

Algorithm:**Code:**

#Encryption:

cipher_text=""

def railfence(plain_text,key):

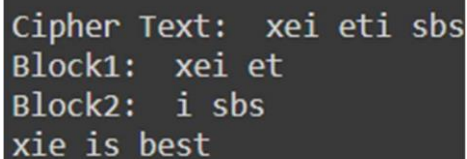
if key==2:

return (plain_text[::2]+plain_text[1::2])

else:

```
        return "number of rails not supported"
cipher_text= railfence("xie is best ", 2)
print(cipher_text)
```

```
#Decryption:
block1=cipher_text[:6:]
print(block1)
block2=cipher_text[6::]
print(block2)
x1=print(block1[0]+block2[0]+block1[1]+block2[1]+block1[2]+block2[2]
]+block1[3]+block2[3]+
    block1[4]+block2[4]+block1[5]+block2[5])
```

Output:

```
Cipher Text: xei eti sbs
Block1: xei et
Block2: i sbs
xie is best
```

2b] Product Cipher Encryption and Decryption using Railfence**Theory:**

In cryptography, a product cipher combines two or more transformations in a manner intending that the resulting cipher is more secure than the individual components to make it resistant to cryptanalysis. The product cipher combines a sequence of simple transformations such

Algorithm:**Code:**

```
#Encryption:
cipher_text=""
def railfence(plain_text,key):
```

```
    if key==2:  
return  
(plain_text[::2]+plain_text[1::2])  
else:  
return "number of rails not  
supported"
```

#Decryption:

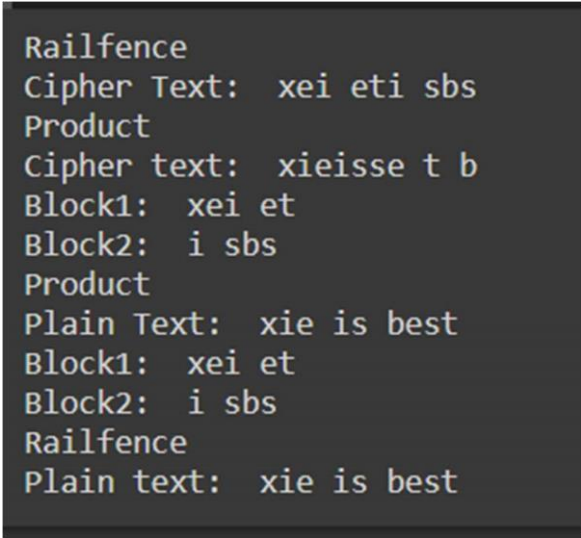
```
def dec(cipher):
```

```
    block1=cipher_text[:6]  
    print("Block1: ",block1)  
    block2=cipher_text[6:]  
    print("Block2: ",block2)
```

```
    x1=(block1[0]+block2[0]+block1[1]+block2[1]+block1[2]+block2[2]+block1[3]+block2[3]+block1[4]+block2[4]+block1[5]+block2[5])  
    return x1
```

```
    cipher_text= railfence("xie is best ", 2)  
    product_c = railfence(cipher_text,2)  
    print("Railfence\nCipher Text: ",cipher_text)  
    print("Product\nCipher text: ",product_c)  
    product_d = dec(product_c)  
    print("Product\nPlain Text: ",product_d)  
    plainText = dec(product_d)  
    print("Railfence\nPlain text: ",plainText)
```

Output:



```
Railfence  
Cipher Text:  xei eti sbs  
Product  
Cipher text:  xieisse t b  
Block1:  xei et  
Block2:  i sbs  
Product  
Plain Text:  xie is best  
Block1:  xei et  
Block2:  i sbs  
Railfence  
Plain text:  xie is best
```

Conclusion: The railfence cipher technique and vigenere cipher technique is executed successfully along with the code and algorithm.

