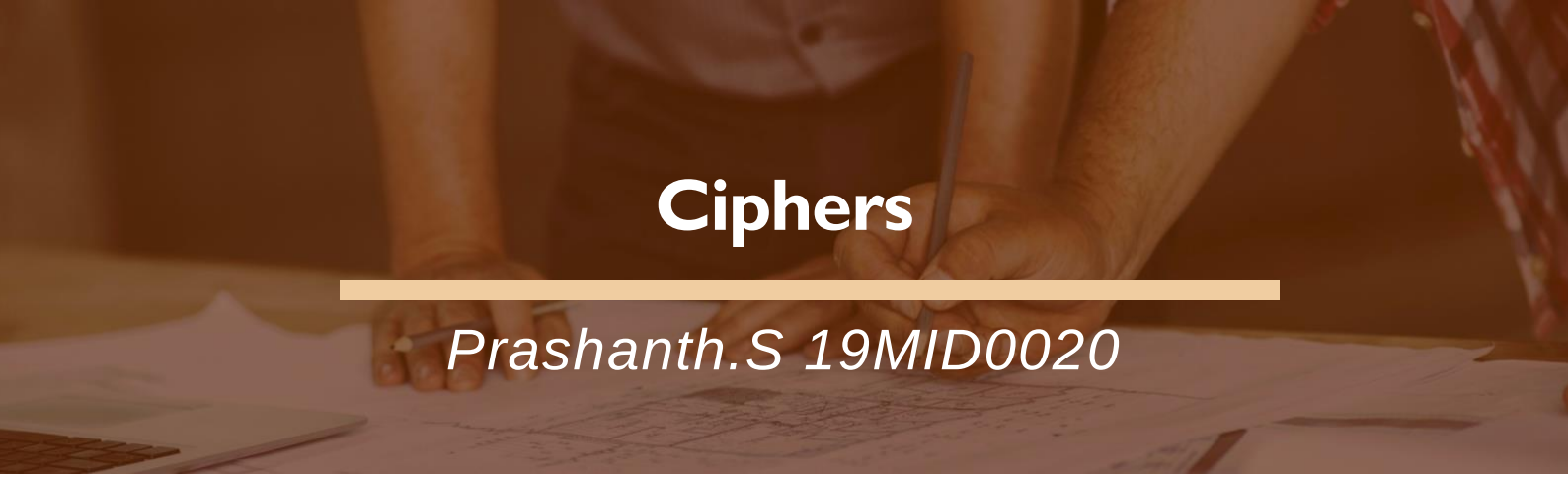

Ciphers

Prashanth.S 19MID0020

Ceaser Cipher

In [1]:
```
def a2d(text):
    return [ord(i) for i in text]
```

In [2]:
```
def encrpyt(text, key):
    dtext = a2d(text)
    result = []
    for i in dtext:
        if (i >= 65 and i <= 90):
            result.append((((i - 65) + key) % 26) + 65)
        elif(i >= 97 and i <= 122):
            result.append((((i - 97) + key) % 26) + 97)
        else:
            result.append(i)
    final = list(map(chr, result))
    return ''.join(final)
```

In [3]:
```
def decrypt(text, key):
    dtext = a2d(text)
    result = []
    for i in dtext:
        if (i >= 65 and i <= 90):
            result.append((((i - 65) - key) % 26) + 65)
        elif(i >= 97 and i <= 122):
            result.append((((i - 97) - key) % 26) + 97)
        else:
            result.append(i)
    final = list(map(chr, result))
    return ''.join(final)
```

In [4]:
```
if __name__ == '__main__':
    text = input("Before encryption Plain text: ")
    key = int(input("Key: "))

    cipher = encrpyt(text, key)
    print("Cipher text : {}".format(cipher))

    plain = decrypt(cipher, key)
    print("After decryption Plain text : {}".format(plain))
```

```
Before encryption Plain text: prashanth
Key: 23
Cipher text : moxpexkqe
After decryption Plain text : prashanth
```

Play-fair Cipher

In [1]:
```
import string
from collections import OrderedDict
import numpy as np
from ordered_set import OrderedSet
```

In [2]:
```
def key_text_rule(key):
    for j in range(len(key)):
        for i in range(len(key)):
            if ((i%2==0) and (i+1!=len(key))):
                if ((key[i]) == (key[i+1])):
                    near = i+1
                    key = key[:near] + 'x' + key[near:]
                    break

    if (len(key)%2!=0):
        key = key[:len(key)+1] + 'z'
        return key
    else:
        return key
```

In [3]:
```
def matrix_fill(key):
    key = "".join(OrderedDict.fromkeys(key))  ## remove the repeated characters in the string
    str1 = string.ascii_lowercase
    for i in key:
        if i in str1:
            str1 = str1.replace(i,'')
    str1 = str1.replace('j','')
    matrix_elements = key + str1

    list1 = []
    ind = 0
    for i in range(5):
        temp = []
        for j in range(5):
            temp.append(matrix_elements[ind])
            ind+=1
        list1.append(temp)

    return list1
```

In [4]:
```
key = key_text_rule('monarchy')
plain_text = key_text_rule('instruments')

print(key)
print(plain_text)
```

```
monarchy
instrumentsz
```

In [5]:
```
matrix = matrix_fill(key)
matrix
```

Out[5]:
```
[['m', 'o', 'n', 'a', 'r'],
 ['c', 'h', 'y', 'b', 'd'],
 ['e', 'f', 'g', 'i', 'k'],
 ['l', 'p', 'q', 's', 't'],
 ['u', 'v', 'w', 'x', 'z']]
```

Encryption

In [6]:
```
def same_row_encrypt(ind1,ind2,ind3,ind4,matrix): ## Same 1st index(i.e i)

    ## loop
    if (ind2==4 or ind4==4):
        if (ind2==4):
            ind2 = 0
            print(matrix[ind1][ind2])
            print(matrix[ind3][ind4+1])

        if (ind4==4):
            ind4 = 0
            print(matrix[ind1][ind2+1])
            print(matrix[ind3][ind4])

    ## not a loop
    else:
        print(matrix[ind1][ind2+1])
        print(matrix[ind3][ind4+1])
```

In [7]:
```
def same_col_encrypt(ind1,ind2,ind3,ind4,matrix): ## Same 2nd index(i.e j)

    ## loop
    if (ind1==4 or ind3==4):
        if (ind1==4):
            ind1 = 0
            print(matrix[ind1][ind2])
            print(matrix[ind3+1][ind4])

        if (ind3==4):
            ind3 = 0
            print(matrix[ind1+1][ind2])
            print(matrix[ind3][ind4])

    ## not a loop
    else:
        print(matrix[ind1+1][ind2])
        print(matrix[ind3+1][ind4])
```

In [8]:
```
def diff(ind1,ind2,ind3,ind4,matrix):  ## Not in same row and same column
    print(matrix[ind1][ind4])
    print(matrix[ind3][ind2])
```

In [9]:
```
def check(i_index, j_index, matrix):
    for ind in range(len(i_index)):
        if ((ind%2==0) and ind!=len(i_index)):

            if (i_index[ind]==i_index[ind+1]): ## same i-value
                same_row_encrypt(i_index[ind],j_index[ind],i_index[ind+1],j_index[ind+1],matrix)

            elif (j_index[ind]==j_index[ind+1]): ## same j-value
                same_col_encrypt(i_index[ind],j_index[ind],i_index[ind+1],j_index[ind+1],matrix)

            else:
                diff(i_index[ind],j_index[ind],i_index[ind+1],j_index[ind+1],matrix)
```

In [10]:
```
i_index = []
j_index = []

for k in range(len(plain_text)):
    if (k%2==0) and (k+1!=len(plain_text)):
        word_1 = plain_text[k]
        word_2 = plain_text[k+1]

        ## fiding the letters in the matrix
        for i in range(5):
            for j in range(5):
                if ((word_1==matrix[i][j])):
                    i_index.append(i)
                    j_index.append(j)

        for i in range(5):
            for j in range(5):
                if ((word_2==matrix[i][j])):
                    i_index.append(i)
                    j_index.append(j)

print("Cipher text")
check(i_index, j_index,matrix)
```

```
Cipher text
g
a
t
l
m
z
c
l
r
q
t
x
```

In [11]:
```
def index_fill(plain_text, matrix):

    i_index = []
    j_index = []

    for k in range(len(plain_text)):
        if (k%2==0) and (k+1!=len(plain_text)):
            word_1 = plain_text[k]
            word_2 = plain_text[k+1]
```

```
            ## fiding the letters in the matrix
            for i in range(5):
                for j in range(5):
                    if ((word_1==matrix[i][j])):
                        i_index.append(i)
                        j_index.append(j)

            for i in range(5):
                for j in range(5):
                    if ((word_2==matrix[i][j])):
                        i_index.append(i)
                        j_index.append(j)

    print("Cipher text")
    check(i_index, j_index,matrix)
```

In [12]:
```
diff(2,3,0,2,matrix)     # (i,n) --> (a,g)
same_row_encrypt(3,3,3,4,matrix) # (s,t) --> (t,l)
diff(0,4,4,0,matrix)     # (r,u) --> (m,z)
same_col_encrypt(0,0,2,0,matrix) # (m,e) --> (c,l)
diff(0,2,3,4,matrix)     # (n,t) --> (r,q)
same_col_encrypt(3,3,4,3,matrix) # (s,x) --> (x,a)
```

```
g
a
t
l
m
z
c
l
r
q
x
a
```

Decryption

In [13]:
```
key = key_text_rule('monarchy')
plain_text = key_text_rule('gatlmzclrqtx')

print(key)
print(plain_text)
```

```
monarchy
gatlmzclrqtx
```

In [14]:
```
matrix = matrix_fill(key)
matrix
```

Out[14]:
```
[['m', 'o', 'n', 'a', 'r'],
 ['c', 'h', 'y', 'b', 'd'],
 ['e', 'f', 'g', 'i', 'k'],
 ['l', 'p', 'q', 's', 't'],
 ['u', 'v', 'w', 'x', 'z']]
```

In [15]:
```
def same_row_decrypt(ind1,ind2,ind3,ind4,matrix): ## Same 1st index(i.e i)

    print(end='')

    ## loop
    if (ind2==0 or ind4==0):
        if (ind2==0):
            ind2 = 4
            print(matrix[ind1][ind2])
            print(matrix[ind3][ind4-1])

        if (ind4==0):
            ind4 = 4
            print(matrix[ind1][ind2-1])
            print(matrix[ind3][ind4])

    ## not a loop
    else:
        print(matrix[ind1][ind2-1])
        print(matrix[ind3][ind4-1])
```

In [16]:
```
def same_col_decrypt(ind1,ind2,ind3,ind4,matrix): ## Same 2nd index(i.e j)
    print(end='')

    ## loop
    if (ind1==0 or ind3==0):
        if (ind1==0):
            ind1 = 4
            print(matrix[ind1][ind2])
            print(matrix[ind3-1][ind4])

        if (ind3==0):
            ind3 = 4
            print(matrix[ind1-1][ind2])
            print(matrix[ind3][ind4])

    ## not a loop
    else:
        print(matrix[ind1-1][ind2])
        print(matrix[ind3-1][ind4])
```

In [17]:
```
def diff_decrypt(ind1,ind2,ind3,ind4,matrix):  ## Not in same row and same column
    print(end='')
    print(matrix[ind1][ind4])
    print(matrix[ind3][ind2])
```

In [18]:
```
def check(i_index, j_index, matrix):
    for ind in range(len(i_index)):
        if ((ind%2==0) and ind!=len(i_index)):

            if (i_index[ind]==i_index[ind+1]): ## same i-value
                same_row_decrypt(i_index[ind],j_index[ind],i_index[ind+1],j_index[ind+1],matrix)

            elif (j_index[ind]==j_index[ind+1]): ## same j-value
                same_col_decrypt(i_index[ind],j_index[ind],i_index[ind+1],j_index[ind+1],matrix)

            else:
                diff_decrypt(i_index[ind],j_index[ind],i_index[ind+1],j_index[ind+1],matrix)
```

In [19]:
```
i_index = []
j_index = []

for k in range(len(plain_text)):
    if (k%2==0) and (k+1!=len(plain_text)):
        word_1 = plain_text[k]
        word_2 = plain_text[k+1]

        ## fiding the letters in the matrix
        for i in range(5):
            for j in range(5):
                if ((word_1==matrix[i][j])):
                    i_index.append(i)
                    j_index.append(j)

        for i in range(5):
            for j in range(5):
                if ((word_2==matrix[i][j])):
                    i_index.append(i)
                    j_index.append(j)

print("Plain text")
check(i_index, j_index,matrix)
```

```
Plain text
i
n
s
t
r
u
m
e
n
t
s
z
```

Hill Cipher

In [1]:
```
import string
from collections import OrderedDict
import numpy as np
from ordered_set import OrderedSet
import pymatrix
```

Encryption

In [2]:
```
def key_text_rule(key):
    for j in range(len(key)):
        for i in range(len(key)):
            if ((i%2==0) and (i+1!=len(key))):
                if ((key[i]) == (key[i+1])):
                    near = i+1
                    key = key[:near] + 'x' + key[near:]
                    break

    if (len(key)%2!=0):
        key = key[:len(key)+1] + 'z'
        return key
    else:
        return key
```

In [3]:
```
def text_to_matrix(dict1, text, n):
    list1 = []
    for i in text:
        list1.append(dict1[i])

    matrix = np.array(list1).reshape(n,n)
    return matrix
```

In [4]:
```
def encryption(small_dict, key):
    key_matrix = text_to_matrix(small_dict, key, 2)
    key_matrix = np.matrix(key_matrix)

    main_encrypt_list = []

    for i in range(len(plain_text)):
        plain_list1 = []
        if ((i%2==0) and (i<=len(plain_text))):

            plain_list1.append(small_dict[plain_text[i]])
            plain_list1.append(small_dict[plain_text[i+1]])

            main_encrypt_list.append((np.dot(key_matrix, np.array(plain_list1).reshape(2,))) % 26)

    cipher_text = []
    dict_keys=list(small_dict.keys())

    for i in main_encrypt_list:

        val1 = i[0,0]
        val2 = i[0,1]

        cipher_text.append(dict_keys[val1])
        cipher_text.append(dict_keys[val2])

    cipher_text = ''.join(map(str,cipher_text))
    return (cipher_text, key_matrix)
```

Decryption

In [5]:
```
def gcd(a, b):

    if(b == 0):
        return a
    else:
        return gcd(b, a % b)
```

In [6]:
```
def modulo_multiplicative_inverse(key_matrix_det):

    if (gcd(key_matrix_det,26)==1):

        if (key_matrix_det>27):
            key_matrix_det = key_matrix_det%26

        num = 1
        while((key_matrix_det * num) % 26 !=1):
            num+=1

        return num

    else:
        return 0 # GCD(det,26)!=1, then modulo multiplicative inverse --> Not found
```

In [7]:
```
def decryption(cipher_text, key_matrix):

    key_matrix_det = int(np.linalg.det(key_matrix))
    one_by_det = modulo_multiplicative_inverse(key_matrix_det)

    if one_by_det:
        adj = (pymatrix.matrix(key_matrix.tolist())).adjoint()
        ## converting into numpy and int array

        key_matrix_adj = []
        for i in range(n):
            key_matrix_adj.append(adj[i])

        key_matrix_adj = np.array(key_matrix_adj).astype(int)
        key_matrix_adj

        for i in key_matrix_adj:
            if (i[0] < 0) : i[0] += 26
            if (i[1] < 0) : i[1] += 26

        key_inverse = key_matrix_adj * one_by_det

        main_decrypt_list = []
        for i in range(len(cipher_text)):

            plain_list1 = []
            if ((i%2==0) and (i<=len(cipher_text))):

                plain_list1.append(small_dict[cipher_text[i]])
                plain_list1.append(small_dict[cipher_text[i+1]])

                main_decrypt_list.append(np.round(np.dot(key_inverse, np.array(plain_list1).reshape(2,))) % 26)

        main_decrypt_list = np.int_(main_decrypt_list)

        original_text = []

        for i in main_decrypt_list:
            dict_keys=list(small_dict.keys())
            original_text.append(dict_keys[i[0]])
            original_text.append(dict_keys[i[1]])

        original_text = ''.join(map(str,original_text))

        return original_text

    else:
        return "GCD!=1, No Modulo Multiplicative Inverse"
```

```
In [9]: if __name__ == '__main__':

            small_dict = dict()
            for index, letter in enumerate(string.ascii_lowercase):
                   small_dict[letter] = index + 0

            plain_text = key_text_rule('prashanth')
            print(plain_text)
            n = 2
            key = 'test'

            cipher_text, key_matrix = encryption(small_dict, key)
            plain_text_dec = decryption(cipher_text,key_matrix)

        prashanthz

In [10]: cipher_text

Out[10]: 'pvuedwlxzd'

In [11]: plain_text_dec

Out[11]: 'prashanthz'
```

My codes and assignment document
https://github.com/PrashanthSingaravelan/winter_semester-2022-assignments/tree/main/CSI3002%20Applied%20Cryptography%20and%20Network%20Security/lab%20assignment/assignment_1