Symmetric Cryptographic Algorithms

`Symmetric cryptographic algorithms are the algorithms used in cryptography where the same key is used for the process of both Encryption of plaintext [ek(m) = c] and Decryption of ciphertext [dk(c) = m].`

Commonly Known Algorithms For Demonstration:

- Caesar Cipher
- Shift Cipher
- Substitution Cipher
- Vigenere Cipher
- One Time Pad Cipher
- Rail Fence Cipher
- Permutation Cipher
- Playfair Cipher

Importing required modules

In [1]:

```python
# for handling arrays
import numpy as np

# for providing random sequences
import random as r
```

Inputting Plain Text For Testing out Programs

In [2]:

```python
inputText = input("Enter the plain text: ")

# plain text processed below removing any non alpha characters
text = np.array([])
for iT in inputText:
    if iT.isalpha():
        text = np.append(text, iT.lower())
text = text
```

```
Enter the plain text: Happiness is everything!!
```

Caesar Cipher

In cryptography, a Caesar cipher is categorized as a substitution cipher in which the alphabet in the plain text is shifted by 3 down the alphabet.

In [3]:

```python
def caesarCipher (text):

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in text:
        c = ((ord(t) - 97 + 3) % 26) + 65
        c = chr(c).upper()
        cipherArr = np.append(cipherArr, c)
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for c in cipherArr:
        p = ((ord(c) - 65 - 3) % 26) + 97
        p = chr(p).lower()
        plainArr = np.append(plainArr, p)
    plainText = "".join(plainArr)

    # print output
    print("\033[1mCaesar Cipher\033[0m")
    print("Key: " + "3")
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [4]:

```python
# function call
caesarCipher(text)
```

```
Caesar Cipher
Key: 3
Cipher Text: KDSSLQHVVLVHYHUBWKLQJ
Plain Text: happinessiseverything
```

Shift Cipher

A shift cipher involves replacing each letter in the message by a letter that is some fixed number of positions further along in the alphabet.

In [5]:

```python
def shiftCipher (text, key):

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in text:
        c = ((ord(t) - 97 + key) % 26) + 65
        c = chr(c).upper()
        cipherArr = np.append(cipherArr, c)
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for c in cipherArr:
        p = ((ord(c) - 65 - key) % 26) + 97
        p = chr(p).lower()
        plainArr = np.append(plainArr, p)
    plainText = "".join(plainArr)

    # print output
    print("\n\033[1mShift Cipher\033[0m")
    print("Key: " + str(key))
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [6]:

```python
# preparing the key
key = input("Enter a number (1-25): ")
while not (key.isdigit() and int(key) in range(1,26)) :
    print("Wrong Input!!")
    key = input("Enter a number (1-25): ")
key = int(key)

# function call
shiftCipher(text, key)
```

```
Enter a number (1-25): 7

Shift Cipher
Key: 7
Cipher Text: OHWWPULZZPZLCLYFAOPUN
Plain Text: happinessiseverything
```

Substitution Cipher

Substitution cipher, a data encryption scheme in which units of the plaintext (generally single letters or pairs of letters of ordinary text) are replaced with other symbols or groups of symbols.

In [7]:

```python
def substitutionCipher (text, key):

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in text:
        c = ord(t) - 97
        val = key[c]
        c = chr(val + 65)
        cipherArr = np.append(cipherArr, c)
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for c in cipherArr:
        p = ord(c) - 65
        pos = int(np.where(key==p)[0])
        p = chr(pos + 97)
        plainArr = np.append(plainArr, p)
    plainText = "".join(plainArr)

    # print output
    print("\033[1mSubstitution Cipher\033[0m ")
    print("Key: " + str(key))
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [8]:

```python
# preparing the key
key = np.arange(start = 0, stop = 26)
r.shuffle(key)

# function call
substitutionCipher(text, key)
```

```
Substitution Cipher
Key: [ 3 19  8 11  5 13  9 24 23  1 16  0 22 18  2  6 17 20  4 10  7 12 25 15
 21 14]
Cipher Text: YDGGXSFEEXEFMFUVKYXSJ
Plain Text: happinessiseverything
```

Vigenere Cipher

Vigenère cipher, a type of substitution cipher used for data encryption in which the original plaintext structure is somewhat concealed in the ciphertext by using several different monoalphabetic substitution ciphers rather than just one; the code key specifies which particular substitution is to be employed for encrypting each plaintext symbol.

In [9]:

```python
def vigenereCipher (text, blockSize, key):

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in range(0, len(text), blockSize):
        cBlock = text[t : (t + blockSize)]
        for cB in range(len(cBlock)):
            c = chr(((ord(cBlock[cB]) - 97 + key[cB]) % 26) + 65)
            cipherArr = np.append(cipherArr, c)
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for c in range(0, len(cipherArr), blockSize):
        pBlock = cipherArr[c : (c + blockSize)]
        for pB in range(len(pBlock)):
            p = chr(((ord(pBlock[pB]) - 65 - key[pB]) % 26) + 97)
            plainArr = np.append(plainArr, p)
    plainText = "".join(plainArr)

    # print output
    print("\n\033[1mVigenere Cipher\033[0m")
    print("Key: (" + str(blockSize) + ", " + str(key) + ")")
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [10]:

```python
# getting size of string to input
blockSize = input("Enter a number: ")
while not (blockSize.isdigit()) :
    print("Wrong Input!!")
    blockSize = input("Enter a number: ")
blockSize = int(blockSize)

# getting the key from user
keyText = input("Enter a string of alphabets of size " + str(blockSize) + ": ")
while not (keyText.isalpha() and len(keyText) == blockSize) :
    print("Wrong Input!!")
    keyText = input("Enter a string of alphabets of size " + str(blockSize) + ": ")
keyText = keyText.lower()

# preparing the key
key = np.array([])
for kT in keyText:
    kT = ord(kT) - 97
    key = np.append(key, kT)
key = key.astype(int)

# function call
vigenereCipher(text, blockSize, key)
```

```
Enter a number: 4
Enter a string of alphabets of size 4: qmdl

Vigenere Cipher
Key: (4, [16 12  3 11])
Cipher Text: XMSAYZHDIUVPLQUJJTLYW
Plain Text: happinessiseverything
```

One Time Pad Cipher

In cryptography, a one-time pad is a system in which a randomly generated private key is used only once to encrypt a message that is then decrypted by the receiver using a matching one-time pad and key.

In [11]:

```python
def oneTimePadCipher (text, key) :

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in range(len(text)) :
        c = chr(((ord(text[t]) - 97 + key[t]) % 26) + 65)
        cipherArr = np.append(cipherArr, c)
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for c in range(len(cipherArr)) :
        p = chr(((ord(cipherArr[c]) - 65 - key[c]) % 26) + 97)
        plainArr = np.append(plainArr, p)
    plainText = "".join(plainArr)

    # print output
    print("\033[1mOne Time Pad Cipher\033[0m")
    print("Key: " + str(key))
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [12]:

```python
# preparing the key
key = np.array([])
for i in range(len(text)) :
    key = np.append(key, r.randrange(26))
key = key.astype(int)

# function call
oneTimePadCipher(text, key)
```

```
One Time Pad Cipher
Key: [ 0  1 17 11 16  3 16 24  3 18 16 21 13 13 11 25  2  9  2 21  5]
Cipher Text: HBGAYQUQVAIZIRCXVQKIL
Plain Text: happinessiseverything
```

Rail Fence Cipher

Rail fence cipher is a type of transposition cipher where the letters are not changed, but only switched around regarding their positioning in the message using a matrix.

In [13]:

```python
def railFenceCipher (textArr, key):

    # converting inputText into cipherText
    cipherArr = np.transpose(textArr)
    cipherText = ""
    for c in range(len(cipherArr)):
        cipherText += "".join(cipherArr[c])

    # converting cipherText into plainText
    plainArr = np.transpose(cipherArr)
    plainText = ""
    for p in range(len(plainArr)):
        plainText += "".join(plainArr[p])

    # print output
    print("\n\033[1mRail Fence Cipher\033[0m")
    print("Key: " + str(key))
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [14]:

```python
# preparing the key
key = input("Enter the depth (d <= " + str(int(len(text)/2)) + "): ")
while not (key.isdigit() and int(key) <= int(len(text)/2)) :
    print("Wrong Input!!")
    key = input("Enter the depth (d <= " + str(int(len(text)/2)) + "): ")
key = int(key)

# preparing the textArr
if len(text) % key != 0:
    textArr = np.append(text, np.repeat(" ", (key - (len(text) % key))))
    textArr = textArr.reshape(key, -1)

# function call
railFenceCipher(textArr, key)
```

```
Enter the depth (d <= 10): 4

Rail Fence Cipher
Key: 4
Cipher Text: heviasenpsrgpiy ist neh
Plain Text: happinessiseverything
```

Permutation Cipher (error: problem in for loop indexing)

Permutation cipher is a type of transposition cipher where the text is split into substrings and each substring is switched around in a certain order.

In [15]:

```python
def permutationCipher (textArr, blockSize, key):

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in range(0, len(textArr), blockSize):
        cBlock = textArr[t: (t + blockSize)]
        for c in range(len(cBlock)):
            cipherArr = np.append(cipherArr, cBlock[key[c]])
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for c in range(0, len(cipherArr), blockSize):
        pBlock = cipherArr[c: (c + blockSize)]
        for p in range(len(pBlock)):
            for k in range(len(key)):
                if p == key[k]:
                    plainArr = np.append(plainArr, pBlock[k])
    plainText = "".join(plainArr)

    # print output
    print("\n\033[1mPermutation Cipher\033[0m")
    print("Key: (" + str(blockSize) + ", " + str(key) + ")")
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

Preparations for Function

In [16]:

```python
# getting the blockSize
blockSize = input("Enter the Block size: (less than or equal to " + str(len(text)) + "): ")
while not (blockSize.isdigit() and int(blockSize) <= len(text)):
    print("Wrong Input!!")
    blockSize = input("Enter the Block size: (less than or equal to " + str(len(text)) + "): ")
blockSize = int(blockSize)

# preparing the key
key = np.array(range(blockSize))
r.shuffle(key)

# preparing the textArr
if len(text) % blockSize != 0:
    textArr = np.append(text, np.repeat(" ", (blockSize - (len(text) % blockSize))))

# function call
permutationCipher(textArr, blockSize, key)
```

```
Enter the Block size: (less than or equal to 21): 17

Permutation Cipher
Key: (17, [ 6 10  0  1  9 13  5  8  3  2 15  4  7 11 16 12 14])
Cipher Text: eshaiensppyisetvr  hi    gn
Plain Text: happinessiseverything
```

Playfair Cipher

Playfair cipher, a cipher involving a digraphic substitution from a single alphabet square which begins with the letters of a keyword and continues with the remaining letters of the alphabet less J.

In [17]:

```python
def playfairCipher (textArr, key):

    # converting inputText into cipherText
    cipherArr = np.array([])
    for t in range(0, len(textArr), 2):
        flag = 0
        t1 = textArr[t]
        t2 = textArr[t+1]

        for i in range(len(key)):
            for j in range(len(key[0])):
                if t1 == key[i][j]:
                    flag += 1
                    pos1 = [i,j]
                elif t2 == key[i][j]:
                    flag += 1
                    pos2 = [i,j]
                if flag == 2:
                    break
        if pos1[0] == pos2[0]:
            if pos1[1] == 4:
                pos1[1] = 0
            else:
                pos1[1] += 1
            if pos2[1] == 4:
                pos2[1] = 0
            else:
                pos2[1] += 1
        elif pos1[1] == pos2[1]:
            if pos1[0] == 4:
                pos1[0] = 0
            else:
                pos1[0] += 1
            if pos2[0] == 4:
                pos2[0] = 0
            else:
                pos2[0] += 1
        else:
            pos1[1], pos2[1] = pos2[1], pos1[1]
        t1 = key[pos1[0]][pos1[1]]
        t2 = key[pos2[0]][pos2[1]]
        cipherArr = np.append(cipherArr, [t1, t2])
    cipherText = "".join(cipherArr)

    # converting cipherText into plainText
    plainArr = np.array([])
    for t in range(0, len(cipherArr), 2):
        flag = 0
        c1 = cipherArr[t]
        c2 = cipherArr[t+1]
        for i in range(len(key)):
            for j in range(len(key[0])):
                if c1 == key[i][j]:
                    flag -= 1
                    pos1 = [i,j]
                elif c2 == key[i][j]:
                    flag -= 1
                    pos2 = [i,j]
                if flag == 2:
                    break
        if pos1[0] == pos2[0]:
            if pos1[1] == 0:
                pos1[1] = 4
            else:
                pos1[1] -= 1
            if pos2[1] == 0:
                pos2[1] = 4
            else:
                pos2[1] -= 1
        elif pos1[1] == pos2[1]:
            if pos1[0] == 0:
                pos1[0] = 4
            else:
                pos1[0] -= 1
            if pos2[0] == 0:
                pos2[0] = 4
            else:
                pos2[0] -= 1
        else:
            pos1[1], pos2[1] = pos2[1], pos1[1]
        c1 = key[pos1[0]][pos1[1]]
        c2 = key[pos2[0]][pos2[1]]
        plainArr = np.append(plainArr, [c1, c2])
    plainText = "".join(plainArr)

    # print output
    print("\n\033[1mPlayfair Cipher\033[0m")
    print("Key: " + str(key))
    print("Cipher Text: " + cipherText)
    print("Plain Text: " + plainText)
```

In [18]:

```python
# getting input of a string
keyText = input('Enter a string: ')
while not (keyText.isalpha() and len(keyText) != 0) :
    print("Wrong Input!!")
    keyText = input("Enter a word: ")

# preparing the key
key = np.array([])
for kT in keyText:
    kT = kT.lower()
    if kT.isalpha() and (kT not in key) and (kT != 'j') :
            key = np.append(key, kT)
i = 97
while i < 123:
    if chr(i) not in key and chr(i) != 'j':
        key = np.append(key, chr(i))
    i += 1
key = np.reshape(key, (5,5))

# preparing the textArr
textArr = np.array([])
for t in range(len(text)):
    textArr = np.append(textArr, text[t])
    if t != (len(text) - 1):
        if text[t] == text[t+1] :
            textArr = np.append(textArr, 'x')
if len(textArr) % 2 != 0 :
    textArr = np.append(textArr, 'x')

# function call
playfairCipher(textArr, key)
```

```
Enter a string: happy

Playfair Cipher
Key: [['h' 'a' 'p' 'y' 'b']
 ['c' 'd' 'e' 'f' 'g']
 ['i' 'k' 'l' 'm' 'n']
 ['o' 'q' 'r' 's' 't']
 ['u' 'v' 'w' 'x' 'z']]
Cipher Text: apywhllgxyomrfwdspobkifz
Plain Text: hapxpinesxsiseverythingx
```

```
<ipython-input-18-5a8ad91ebff6>:11: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if kT.isalpha() and (kT not in key) and (kT != 'j') :
```
